ExtendedDateTool

Initial Rough Version

ExtendedDateTool handles requirements for expressing a specified date relative to now as a string in a "friendly format" (e.g. "3 minutes ago", "tomorrow", "3 days from now")

Below is the code for a first run through at making a tool to cover these requirements. This version is still very rough around the edges.

Code consists of:

- A tool for the toolbox

```
/*
 * Copyright 2003-2004 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.apache.velocity.tools.generic;

import java.util.Calendar;
import org.apache.velocity.tools.generic.DateTool;

/**
 * Extension to Velocity DateTool which provides methods for expressing
 * Calendar objects in a human-friendly, relative format
 *
 * @author <a href="mailto:c.townson@nature.com">Christopher Townson</a>
 *
 * TODO externalize unit conversion methods to dedicated tool?
 * TODO better handling of strings/pluralisation/i8n?
 * TODO integration of properties with velocity.properties
 * TODO clean up problematic handling of dates when time period to be expressed
 *                 is not suited to "friendly-format"
 */
public class ExtendedDateTool extends DateTool
{
        /**
         * number of milliseconds in a second
         */
        public static final long MILLIS_TO_SECOND = 1000;

        /**
         * number of millseconds in a minute
         */
        public static final long MILLIS_TO_MINUTE = 60000;

        /**
         * number of milliseconds in an hour
         */
        public static final long MILLIS_TO_HOUR = 3600000;

        /**
         * number of milliseconds in a day
         */
        public static final long MILLIS_TO_DAY = 86400000;

        /**
         * String to append to dates which are _after_ the current date
```

```
 */
public static final String APPEND_FUTURE =
        Messages.getString("ExtendedDateTool.string.date.append.future");


/**
 * String to append to dates which are _before_ the current date
 */
public static final String APPEND_PAST =
        Messages.getString("ExtendedDateTool.string.date.append.past");

/**
 * Strings for unit: seconds (singular)
 */
public static final String UNIT_SECOND =
        Messages.getString("ExtendedDateTool.string.unit.seconds.singular");

/**
 * String for unit seconds (plural)
 */
public static final String UNIT_SECONDS =
        Messages.getString("ExtendedDateTool.string.unit.seconds.plural");

/**
 * String for unit: minutes (singular)
 */
public static final String UNIT_MINUTE =
        Messages.getString("ExtendedDateTool.string.unit.minutes.singular");

/**
 * String for unit minutes (plural)
 */
public static final String UNIT_MINUTES =
        Messages.getString("ExtendedDateTool.string.unit.minutes.plural");

/**
 * String for unit: minutes (singular)
 */
public static final String UNIT_HOUR =
        Messages.getString("ExtendedDateTool.string.unit.hours.singular");

/**
 * String for unit minutes (plural)
 */
public static final String UNIT_HOURS =
        Messages.getString("ExtendedDateTool.string.unit.hours.plural");

/**
 * String for unit: days (singular)
 */
public static final String UNIT_DAY =
        Messages.getString("ExtendedDateTool.string.unit.days.singular");

/**
 * String for unit days (plural)
 */
public static final String UNIT_DAYS =
        Messages.getString("ExtendedDateTool.string.unit.days.plural");

/**
 * String for unit: 1 day into the future
 */
public static final String YESTERDAY =
        Messages.getString("ExtendedDateTool.string.date.yesterday");

/**
 * String for unit: 1 day into the past
 */
public static final String TOMORROW =
        Messages.getString("ExtendedDateTool.string.date.tomorrow");
```

```
/**
 * String for unit: weeks (singular)
 *
 * Not currently in use as any period over 7 days is better displayed in a
 * standard date format
 */
public static final String UNIT_WEEK =
        Messages.getString("ExtendedDateTool.string.unit.weeks.singular");

/**
 * String for unit: weeks (plural)
 *
 * Currently only used to determine the fact that date should be
 * displayed in a standard format
 */
public static final String UNIT_WEEKS =
        Messages.getString("ExtendedDateTool.string.unit.weeks.plural");

/**
 * Default output format for dates which are too distant to be suitable
 * for rendering in "friendly format"
 */
public static final String DEFAULT_FORMAT =
        Messages.getString("ExtendedDateTOol.string.date.format.default");

/**
 * Returns a provided Date instance in a human-friendly String format
 *
 * @param then The Calendar object to convert to human-friendly format
 * @return The date in a human-friendly format (e.g. 3 days ago)
 */
public String getFriendlyDate(Calendar then)
{
        String friendlyDate = null;
        String append = null;

        // get the difference between the now and then as a long
        long diff = getDifferenceBetweenNowAnd(then);

        // is Calendar in the past or the future?
        if (isPast(diff))
        {
                diff -= (diff * 2);
                append = APPEND_PAST;
        }
        else
        {
                append = APPEND_FUTURE;
        }

        if (isSeconds(diff))
        {
                if (isSingular(diff))
                {
                        friendlyDate = convertMillisToSeconds(diff) +
                        " " + UNIT_SECOND + " " + append;
                }
                else
                {
                        friendlyDate = convertMillisToSeconds(diff) +
                        " " + UNIT_SECONDS + " " + append;
                }
        }
        else if (isMinutes(diff))
        {
                if (isSingular(diff))
                {
                        friendlyDate = convertMillisToMinutes(diff) +
                        " " + UNIT_MINUTE + " " + append;
                }
                else
```

```
                {
                        friendlyDate = convertMillisToMinutes(diff) +
                        " " + UNIT_MINUTES + " " + append;
                }
        }
        else if (isHours(diff))
        {
                if (isSingular(diff))
                {
                        friendlyDate = convertMillisToHours(diff) +
                        " " + UNIT_HOUR + " " + append;
                }
                else
                {
                        friendlyDate = convertMillisToHours(diff) +
                        " " + UNIT_HOURS + " " + append;
                }
        }
        else if (isDays(diff))
        {
                if (isSingular(diff) && append == APPEND_FUTURE)
                {
                        friendlyDate = TOMORROW;
                }
                else if(isSingular(diff) && append == APPEND_PAST)
                {
                        friendlyDate = YESTERDAY;
                }
                else
                {
                        friendlyDate = convertMillisToDays(diff) +
                        " " + UNIT_DAYS + " " + append;
                }
        }
        else
        {
                friendlyDate = super.format(DEFAULT_FORMAT,then.getTime());
        }

        // return friendly string
        return friendlyDate;
}

/**
 * Overloaded getFriendlyDate method for convenience from templates
 *
 * NB. The date format descriptor only applies to the date as it will be
 * returned if it is <strong>not</strong> suitable for friendly format
 *
 * @param format
 * @param obj
 * @return The date in a human-friendly format (e.g. 3 days ago)
 */
public String getFriendlyDate(String format, Object obj) {
        return getFriendlyDate(super.toCalendar(super.toDate(format,obj)));
}

/**
 * Returns a Java timestamp (milliseconds since 1970-01-01)
 *
 * @return long java timestamp for the current date
 */
public long getTimeInMillis()
{
        return Calendar.getInstance().getTimeInMillis();
}

/**
 * Returns a Java timestamp for the supplied Calendar
 *
 * @param cal
```

```
 * @return long the Java timestamp for the supplied calendar
 */
public long getTimeInMillis(Calendar cal)
{
        return cal.getTimeInMillis();
}

/**
 * Returns the time difference between two points expressed in milliseconds
 *
 * @param pointA
 * @param pointB
 * @return The time difference in milliseconds
 */
public long getDifferenceBetween(Calendar pointA, Calendar pointB)
{
        return pointA.getTimeInMillis() - pointB.getTimeInMillis();
}

/**
 * Tests whether the difference between two Calendar instances is
 * a matter of seconds or not
 *
 * @param then The calendar instance for comparison
 * @return The difference in milliseconds between now and supplied Calendar
 *          Returns a positive long if then is after now and a negative long
 *          if then is before now
 */
public long getDifferenceBetweenNowAnd(Calendar then)
{
        return getDifferenceBetween(then, Calendar.getInstance());
}

/**
 * COnvert a duration expressed in milliseconds to seconds
 *
 * @param millis
 * @return milliseconds expressed as seconds
 */
public long convertMillisToSeconds(long millis)
{
        return millis / MILLIS_TO_SECOND;
}

/**
 * Convert a duration expressed in milliseconds to minutes
 *
 * @param millis
 * @return milliseconds expressed as minutes
 */
public long convertMillisToMinutes(long millis)
{
        return millis / MILLIS_TO_MINUTE;
}

/**
 * Convert a duration expressed in milliseconds to hours
 *
 * @param millis
 * @return milliseconds expressed as hours
 */
public long convertMillisToHours(long millis)
{
        return millis / MILLIS_TO_HOUR;
}

/**
 * Convert a duration expressed in milliseconds to days
 *
 * @param millis
 * @return milliseconds expressed as days
```

```
 */
public long convertMillisToDays(long millis)
{
        return millis / MILLIS_TO_DAY;
}

/**
 * Tests for positive or negative result from timestamp comparison
 *
 * @param diff
 * @return whether or not our compared time was in the past
 */
private boolean isPast(long diff)
{
        if (diff < 0)
        {
                return true;
        }
        else
        {
                return false;
        }
}

/**
 * Returns the appropriate time unit string for expressing a
 * millsecond duration in a "friendly" date format
 *
 * @param diff
 * @return The time unit name
 */
private String calculateUnit(long diff)
{
        if (diff > 0 && diff < 60000)
        {
                return UNIT_SECONDS;
        }
        else if (diff >= 60000 && diff < 3600000)
        {
                return UNIT_MINUTES;
        }
        else if (diff >= 3600000 && diff < 86400000)
        {
                return UNIT_HOURS;
        }
        else if (diff >= 86400000 && diff < 604800000)
        {
                return UNIT_DAYS;
        }
        else if (diff >= 604800000)
        {
                return UNIT_WEEKS;
        }
        else
        {
                // default
                return null;
        }
}

/**
 * Returns true if the time unit string is singular; false if the string
 * needs to be pluralised
 *
 * This method will only work on positive longs.
 *
 * @param diff
 * @return Whether or not to pluralise the time unit string
 */
private boolean isSingular(long diff)
{
```

```
        // run this test now to save repeating in conditionals
        String units = calculateUnit(diff);

        if (units == UNIT_SECONDS && diff < 2000)
        {
                return true;
        }
        else if (units == UNIT_MINUTES && diff < 120000)
        {
                return true;
        }
        else if (units == UNIT_HOURS && diff < 7200000)
        {
                return true;
        }
        else if (units == UNIT_DAYS && diff < 172800000)
        {
                return true;
        }
        else if (units == UNIT_WEEKS && diff < 1209600000)
        {
                return true;
        }
        else
        {
                // default
                return false;
        }
}

/**
 * Do we want to talk in seconds?
 *
 * @param diff
 * @return Whether we are dealing with seconds or not
 */
private boolean isSeconds(long diff)
{
        if (calculateUnit(diff) == UNIT_SECONDS)
        {
                return true;
        }
        else
        {
                return false;
        }
}

/**
 * Do we want to talk in minutes?
 *
 * @param diff
 * @return Whether we are dealing with minutes or not
 */
private boolean isMinutes(long diff)
{
        if (calculateUnit(diff) == UNIT_MINUTES)
        {
                return true;
        }
        else
        {
                return false;
        }
}

/**
 * Do we want to talk in hours?
 *
 * @param diff
 * @return Whether we are dealing with hours or not
```

```
         */
        private boolean isHours(long diff)
        {
                if (calculateUnit(diff) == UNIT_HOURS)
                {
                        return true;
                }
                else
                {
                        return false;
                }
        }

        /**
         * Do we want to talk in days?
         *
         * @param diff
         * @return Whether we are dealing with days or not
         */
        private boolean isDays(long diff)
        {
                if (calculateUnit(diff) == UNIT_DAYS)
                {
                        return true;
                }
                else
                {
                        return false;
                }
        }

        /**
         * Do we want to talk in weeks?
         *
         * @param diff
         * @return Whether we are dealing with weeks or not
         */
        private boolean isWeeks(long diff)
        {
                if (calculateUnit(diff) == UNIT_WEEKS)
                {
                        return true;
                }
                else
                {
                        return false;
                }
        }
}
```

- A default Eclipse messages handler

```
/**
 *
 */
package org.apache.velocity.tools.generic;

import java.util.MissingResourceException;
import java.util.ResourceBundle;

/**
 * Eclipse auto-generated helper class to retrieve localised Strings
 * for ExtendedDateTool
 *
 * @author Christopher Townson
 */
public class Messages {
        /**
         * The bundle name
         */
        private static final String BUNDLE_NAME = "org.apache.velocity.tools.generic.messages"; //$NON-NLS-1$

        /**
         * The resource bundle
         */
        private static final ResourceBundle RESOURCE_BUNDLE = ResourceBundle.getBundle(BUNDLE_NAME);

        /**
         * Default constructor
         *
         */
        private Messages() {
                // do nothing
        }

        /**
         * Accessor method for retrieving String properties
         *
         * @param key The key for the String property to get
         * @return The string property
         */
        public static String getString(String key) {
                // TODO Auto-generated method stub
                try {
                        return RESOURCE_BUNDLE.getString(key);
                } catch (MissingResourceException e) {
                        return '!' + key + '!';
                }
        }
}
```

- A messages.properties file

```
ExtendedDateTool.string.date.append.future=from now
ExtendedDateTool.string.date.append.past=ago
ExtendedDateTool.string.unit.seconds.singular=second
ExtendedDateTool.string.unit.seconds.plural=seconds
ExtendedDateTool.string.unit.minutes.singular=minute
ExtendedDateTool.string.unit.minutes.plural=minutes
ExtendedDateTool.string.unit.hours.singular=hour
ExtendedDateTool.string.unit.hours.plural=hours
ExtendedDateTool.string.unit.days.singular=day
ExtendedDateTool.string.unit.days.plural=days
ExtendedDateTool.string.date.yesterday=yesterday
ExtendedDateTool.string.date.tomorrow=tomorrow
ExtendedDateTool.string.unit.weeks.singular=week
ExtendedDateTool.string.unit.weeks.plural=weeks
ExtendedDateTool.string.date.format.default=yyyy-MM-dd HH:mm:ss
```

- A toolbox.xml entry

```
<!-- ExtendedDateTool: additional date manipulation methods -->
<tool>
        <key>xDateTool</key>
        <scope>request</scope>
        <class>org.apache.velocity.tools.generic.ExtendedDateTool</class>
</tool>
```

Refactored Version

Nathan has done a lot of refactoring on that initial rough version which has resulted in the following code:

```
/*
 * Copyright 2006 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.apache.velocity.tools.generic;

import java.util.Calendar;
import java.util.MissingResourceException;
import java.util.ResourceBundle;
import org.apache.velocity.tools.generic.DateTool;

/**
 * Extension to Velocity DateTool which provides methods for expressing
 * date objects in a human-friendly, relative format
 *
 * @author <a href="mailto:c.townson@nature.com">Christopher Townson</a>
 */
public class ExtendedDateTool extends DateTool
{
    /** The number of milliseconds in a second. */
    public static final long MILLIS_PER_SECOND = 1000;

    /** The number of millseconds in a minute. */
    public static final long MILLIS_PER_MINUTE = 60 * MILLIS_PER_SECOND;

    /** The number of milliseconds in an hour. */
    public static final long MILLIS_PER_HOUR = 60 * MILLIS_PER_MINUTE;

    /** The number of milliseconds in a day. */
    public static final long MILLIS_PER_DAY = 24 * MILLIS_PER_HOUR;

    /** The number of milliseconds in a week. */
    public static final long MILLIS_PER_WEEK = 7 * MILLIS_PER_DAY;

    /** An approximation of the number of milliseconds in a month. */
    public static final long MILLIS_PER_MONTH = 30 * MILLIS_PER_DAY;

    /** An approximation of the number of milliseconds in a year. */
    public static final long MILLIS_PER_YEAR = 365 * MILLIS_PER_DAY;


    /** Array of all "millis per X" values. */
```

```
protected static final long[] RELATIVE_UNITS = new long[] {
                                                    1, // millis per milli
                                                    MILLIS_PER_SECOND,
                                                    MILLIS_PER_MINUTE,
                                                    MILLIS_PER_HOUR,
                                                    MILLIS_PER_DAY,
                                                    MILLIS_PER_WEEK,
                                                    MILLIS_PER_MONTH,
                                                    MILLIS_PER_YEAR };

/** Array of all message keys for relative date units. */
protected static final String[] RELATIVE_UNIT_KEYS = new String[] {
                                                    "date.milliseconds",
                                                    "date.seconds",
                                                    "date.minutes",
                                                    "date.hours",
                                                    "date.days",
                                                    "date.weeks",
                                                    "date.months",
                                                    "date.years" };


private static final String DEFAULT_RESOURCE_NAME =
    "org.apache.velocity.tools.generic.messages";

private ResourceBundle textResources;


protected String getText(String key)
{
    if (textResources == null)
    {
        textResources =
            ResourceBundle.getBundle(DEFAULT_RESOURCE_NAME, getLocale());
    }

    try
    {
        return textResources.getString(key);
    }
    catch (MissingResourceException e)
    {
        return '!' + key + '!';
    }
}

/**
 * Returns the value of the largest unit difference between the specified
 * date and the result of getCalendar() in a human-friendly String format.
 *
 * @param date The date to convert to human-friendly format
 * @return the diff between the specified date and "now" in a
 *         human-friendly format (e.g. 3 days ago)
 */
public String toRelativeString(Object date)
{
    String friendly = toRelativeString(date, 1);

    // check for the corner case of "1 day ago" or "1 day from now"
    // and convert those to "yesterday" or "tomorrow"
    if (friendly != null && friendly.startsWith("1 "+getText("date.days.singular")))
    {
        if (friendly.endsWith(getText("date.direction.past")))
        {
            return getText("date.days.singular.past");
        }
        else
        {
            return getText("date.days.plural.future");
        }
    }
```

```
    return friendly;
}

/**
 * Returns the difference between the specified date
 * and the result of getCalendar() in a human-friendly String format
 * with up to the specified number of units.
 *
 * @param date The date to convert to human-friendly format
 * @param maxUnitDepth The maximum number of units deep to show
 * @return the diff between the specified date and "now" in a
 *         human-friendly format (e.g. 3 days 4 hours 2 seconds ago)
 */
public String toRelativeString(Object date, int maxUnitDepth)
{
    return toRelativeString(date, getCalendar(), maxUnitDepth);
}

/**
 * Returns the value of the largest unit difference between the specified
 * dates in a human-friendly String format.
 *
 * @param dateA The primary date to be compared
 * @param dateB The secondary date to be compared
 * @return the diff between the specified date and "now" in a
 *         human-friendly format (e.g. 3 days ago)
 */
public String toRelativeString(Object dateA, Object dateB)
{
    return toRelativeString(dateA, dateB, 1);
}

/**
 * Returns the value of the largest unit difference between the specified
 * dates in a human-friendly String format with up to the specified number
 * of units.
 *
 * @param dateA The primary date to be compared
 * @param dateB The secondary date to be compared
 * @param maxUnitDepth The maximum number of units deep to show
 * @return the diff between the specified date and "now" in a
 *         human-friendly format (e.g. 3 days 4 hours 2 seconds ago)
 */
public String toRelativeString(Object dateA, Object dateB,
                               int maxUnitDepth)
{
    // get the difference between the date and now
    Long difference = diff(dateA, dateB);
    if (difference == null)
    {
        return null;
    }
    long diff = difference.longValue();

    // is the dateA before or after dateB
    boolean isPast = (diff < 0);
    if (isPast)
    {
        // work only with future values
        diff *= -1;
    }

    // get the positive friendly value
    String friendly = toRelativeString(diff, maxUnitDepth);

    // then add the direction
    if (isPast)
    {
        friendly += " " + getText("date.direction.past");
    }
    else
```

```
    {
        friendly += " " + getText("date.direction.future");
    }
    return friendly;
}

/**
 * Converts the specified duration of milliseconds into larger units up to
 * the specified number of positive units, beginning with the largest
 * positive unit.  e.g.
 * <code>toRelativeString(181453, 3)</code> will return
 * "3 minutes 1 second 453 milliseconds",
 * <code>toRelativeString(181453, 2)</code> will return
 * "3 minutes 1 second", and
 * <code>toRelativeString(180000, 2)</code> will return
 * "3 minutes".
 */
protected String toRelativeString(long diff, int maxUnitDepth)
{
    if (diff < 0)
    {
        return null;
    }
    if (diff == 0)
    {
        //TODO? should we differentiate between "now" and "same time"?
        return getText("date.now");
    }

    long value = 0;
    long remainder = 0;
    String unitKey = null;

    // determine the largest unit and calculate the value and remainder
    for (int i = 0; i < RELATIVE_UNITS.length + 1; i++)
    {
        // if diff < MILLIS_PER_MINUTE or we're at the end
        if (diff < RELATIVE_UNITS[i] || i > RELATIVE_UNITS.length)
        {
            // then we're working with seconds
            value = diff / RELATIVE_UNITS[i - 1];
            remainder = diff - (value * RELATIVE_UNITS[i - 1]);
            unitKey = RELATIVE_UNIT_KEYS[i - 1];
            break;
        }
    }

    // select proper pluralization
    if (value == 1)
    {
        unitKey += ".singular";
    }
    else
    {
        unitKey += ".plural";
    }

    // combine the value and the unit
    String output = value + ' ' + getText(unitKey);

    // recurse over the remainder if it exists and more units are allowed
    if (maxUnitDepth > 1 && remainder > 0)
    {
        output += " " + toRelativeString(remainder, maxUnitDepth - 1);
    }
    return output;
}

/**
 * Returns the time difference between two dates expressed in milliseconds
 *
```

```
     * @param dateA
     * @param dateB
     * @return The time difference in milliseconds
     */
    public Long diff(Object dateA, Object dateB)
    {
        Calendar calA = toCalendar(dateA);
        Calendar calB = toCalendar(dateB);
        if (calA == null || calB == null)
        {
            return null;
        }
        return new Long(calA.getTimeInMillis() - calB.getTimeInMillis());
    }

    /**
     * @param date The date for comparison
     * @return The difference in milliseconds between the supplied date and
     *         the date returned by getCalendar() (i.e. "now"). Returns a
     *         positive long if then is after now and a negative long if then
     *         is before now or <code>null</code> if the parameter can not be
     *         converted to a Calendar.
     */
    public Long diff(Object date)
    {
        return diff(date, getCalendar());
    }

}
```

- And the following updated messages.properties

```
date.direction.future=from now
date.direction.past=ago
date.now=now
date.milliseconds.singular=millisecond
date.milliseconds.plural=milliseconds
date.seconds.singular=second
date.seconds.plural=seconds
date.minutes.singular=minute
date.minutes.plural=minutes
date.hours.singular=hour
date.hours.plural=hours
date.days.singular=day
date.days.singular.past=tomorrow
date.days.singular.future=yesterday
date.days.plural=days
date.weeks.singular=week
date.weeks.plural=weeks
date.months.singular=month
date.months.plural=months
date.years.singular=year
date.years.plural=years
```