DailyRollingFileAppender

Configuration Example

{{

Unknown macro: {"<appender name="FileFileAppender" class="org.apache.log4j.DailyRollingFileAppender"> <param name="append" value="true" /> <param name="encoding" value="UTF-8" /> <param name="file" value="mydogfile.log" /> <param name="DatePattern" value="'.'yyyy-MM-dd'" /> <layout class="org.apache.log4j.PatternLayout"> <param name="ConversionPattern" value="[%-25d{ISO8601}] %-5p %x %C{1} -- %mn" /> </layout>< /appender>}}

}

Custom DailyRollingFileAppender with MaxBackupIndex

I've change the DailyRollingFileAppender to support the MaxBackupIndex, this is the class to add to the jar that contains the log4j library:

```
/*
 * Copyright 1999-2005 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.log4j;

import java.io.IOException;
import java.io.File;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Calendar;
import java.util.TimeZone;
import java.util.Locale;

import org.apache.log4j.helpers.LogLog;
import org.apache.log4j.spi.LoggingEvent;

/**
   DailyMaxRollingFileAppender extends {@link FileAppender} so that the
   underlying file is rolled over at a user chosen frequency.

   <p>The rolling schedule is specified by the <b>DatePattern</b>.
   This pattern should follow the {@link SimpleDateFormat} conventions.
   In particular, you <em>must</em> escape literal text within a pair
   of single quotes. A formatted version of the date pattern is used
   as the suffix for the rolled file name.

   <p>For example, if the <b>File</b> option is set to
   <code>/foo/bar.log</code> and the <b>DatePattern</b> set to
   <code>'.'yyyy-MM-dd</code>, on 2001-02-16 at midnight, the logging
   file <code>/foo/bar.log</code> will be copied to
   <code>/foo/bar.log.2001-02-16</code> and logging for 2001-02-17
   will continue in <code>/foo/bar.log</code> until it rolls over
   the next day.

   <p>Is is possible to specify monthly, weekly, half-daily, daily,
   hourly, or minutely rollover schedules.
```

```
<p><table border="1" cellpadding="2">
<tr>
<th>DatePattern</th>
<th>Rollover schedule</th>
<th>Example</th>

<tr>
<td><code>'.'yyyy-MM</code>
<td>Rollover at the beginning of each month</td>

<td>At midnight of May 31st, 2002 <code>/foo/bar.log</code> will be
copied to <code>/foo/bar.log.2002-05</code>. Logging for the month
of June will be output to <code>/foo/bar.log</code> until it is
also rolled over the next month.

<tr>
<td><code>'.'yyyy-ww</code>

<td>Rollover at the first day of each week. The first day of the
week depends on the locale.</td>

<td>Assuming the first day of the week is Sunday, on Saturday
midnight, June 9th 2002, the file <i>/foo/bar.log</i> will be
copied to <i>/foo/bar.log.2002-23</i>.  Logging for the 24th week
of 2002 will be output to <code>/foo/bar.log</code> until it is
rolled over the next week.

<tr>
<td><code>'.'yyyy-MM-dd</code>

<td>Rollover at midnight each day.</td>

<td>At midnight, on March 8th, 2002, <code>/foo/bar.log</code> will
be copied to <code>/foo/bar.log.2002-03-08</code>. Logging for the
9th day of March will be output to <code>/foo/bar.log</code> until
it is rolled over the next day.

<tr>
<td><code>'.'yyyy-MM-dd-a</code>

<td>Rollover at midnight and midday of each day.</td>

<td>At noon, on March 9th, 2002, <code>/foo/bar.log</code> will be
copied to <code>/foo/bar.log.2002-03-09-AM</code>. Logging for the
afternoon of the 9th will be output to <code>/foo/bar.log</code>
until it is rolled over at midnight.

<tr>
<td><code>'.'yyyy-MM-dd-HH</code>

<td>Rollover at the top of every hour.</td>

<td>At approximately 11:00.000 o'clock on March 9th, 2002,
<code>/foo/bar.log</code> will be copied to
<code>/foo/bar.log.2002-03-09-10</code>. Logging for the 11th hour
of the 9th of March will be output to <code>/foo/bar.log</code>
until it is rolled over at the beginning of the next hour.


<tr>
<td><code>'.'yyyy-MM-dd-HH-mm</code>

<td>Rollover at the beginning of every minute.</td>

<td>At approximately 11:23,000, on March 9th, 2001,
<code>/foo/bar.log</code> will be copied to
<code>/foo/bar.log.2001-03-09-10-22</code>. Logging for the minute
of 11:23 (9th of March) will be output to
<code>/foo/bar.log</code> until it is rolled over the next minute.
```

```
   </table>

   <p>Do not use the colon ":" character in anywhere in the
   <b>DatePattern</b> option. The text before the colon is interpeted
   as the protocol specificaion of a URL which is probably not what
   you want.

   <p>You have also to define the maximum number of file are kept
   before the oldest is erased.</p>

   @author Eirik Lygre
   @author Ceki G&uuml;lc&uuml;
   @author Riccardo Nicosia;
*/
public class DailyMaxRollingFileAppender extends FileAppender
{
  // The code assumes that the following constants are in a increasing
  // sequence.
  static final int TOP_OF_TROUBLE=-1;
  static final int TOP_OF_MINUTE = 0;
  static final int TOP_OF_HOUR   = 1;
  static final int HALF_DAY      = 2;
  static final int TOP_OF_DAY    = 3;
  static final int TOP_OF_WEEK   = 4;
  static final int TOP_OF_MONTH  = 5;

  /**
     The date pattern. By default, the pattern is set to
     "'.'yyyy-MM-dd" meaning daily rollover.
   */
  private String datePattern = "'.'yyyy-MM-dd";

  /**
     There is one backup file by default.
   */
  private int  maxBackupIndex  = 1;

  /**
     The log file will be renamed to the value of the
     scheduledFilename variable when the next interval is entered. For
     example, if the rollover period is one hour, the log file will be
     renamed to the value of "scheduledFilename" at the beginning of
     the next hour.

     The precise time when a rollover occurs depends on logging
     activity.
  */
  private String scheduledFilename;

  /**
     The next time we estimate a rollover should occur. */
  private long nextCheck = System.currentTimeMillis () - 1;

  Date now = new Date();

  SimpleDateFormat sdf;

  RollingPastCalendar rpc = new RollingPastCalendar();

  int checkPeriod = TOP_OF_TROUBLE;

  // The gmtTimeZone is used only in computeCheckPeriod() method.
  static final TimeZone gmtTimeZone = TimeZone.getTimeZone("GMT");

  /**
     The default constructor does nothing. */
  public DailyMaxRollingFileAppender()
  {}

  /**
    Instantiate a <code>DailyRollingFileAppender</code> and open the
```

```
  file designated by <code>filename</code>. The opened filename will
  become the ouput destination for this appender.

  */
public DailyMaxRollingFileAppender (Layout layout, String filename,
                                    String datePattern)
throws IOException
{
  super(layout, filename, true);
  this.datePattern = datePattern;
  activateOptions();
}

/**
   The <b>DatePattern</b> takes a string in the same format as
   expected by {@link SimpleDateFormat}. This options determines the
   rollover schedule.
 */
public void setDatePattern(String pattern) {
  datePattern = pattern;
}

/** Returns the value of the <b>DatePattern</b> option. */
public String getDatePattern() {
  return datePattern;
}

/**
   Set the maximum number of backup files to keep around.

   <p>The <b>MaxBackupIndex</b> option determines how many backup
   files are kept before the oldest is erased. This option takes
   a positive integer value. If set to zero, then there will be no
   backup files and the log file will be renamed to the value of the
   scheduledFilename variable when the next interval is entered.
 */
public void setMaxBackupIndex(int maxBackups)
{
  this.maxBackupIndex = maxBackups;
}

/**
   Returns the value of the <b>MaxBackupIndex</b> option.
 */
public int getMaxBackupIndex() {
  return maxBackupIndex;
}

public void activateOptions()
{
  super.activateOptions();

  LogLog.debug("Max backup file kept: "+ maxBackupIndex + ".");

  if(datePattern != null && fileName != null)
  {
    now.setTime(System.currentTimeMillis());
    sdf = new SimpleDateFormat(datePattern);
    int type = computeCheckPeriod();
    printPeriodicity(type);
    rpc.setType(type);
    File file = new File(fileName);
    scheduledFilename = fileName+sdf.format(new Date(file.lastModified()));
  }
  else
  {
    LogLog.error("Either File or DatePattern options are not set for appender ["
                 +name+"].");
  }
}
```

```
void printPeriodicity(int type)
{
  switch(type) {
  case TOP_OF_MINUTE:
    LogLog.debug("Appender [[+name+]] to be rolled every minute.");
    break;
  case TOP_OF_HOUR:
    LogLog.debug("Appender ["+name
                 +"] to be rolled on top of every hour.");
    break;
  case HALF_DAY:
    LogLog.debug("Appender ["+name
                 +"] to be rolled at midday and midnight.");
    break;
  case TOP_OF_DAY:
    LogLog.debug("Appender ["+name
                 +"] to be rolled at midnight.");
    break;
  case TOP_OF_WEEK:
    LogLog.debug("Appender ["+name
                 +"] to be rolled at start of week.");
    break;
  case TOP_OF_MONTH:
    LogLog.debug("Appender ["+name
                 +"] to be rolled at start of every month.");
    break;
  default:
    LogLog.warn("Unknown periodicity for appender [[+name+]].");
  }
}

// This method computes the roll over period by looping over the
// periods, starting with the shortest, and stopping when the r0 is
// different from from r1, where r0 is the epoch formatted according
// the datePattern (supplied by the user) and r1 is the
// epoch+nextMillis(i) formatted according to datePattern. All date
// formatting is done in GMT and not local format because the test
// logic is based on comparisons relative to 1970-01-01 00:00:00
// GMT (the epoch).

 int computeCheckPeriod()
{
  RollingPastCalendar rollingPastCalendar = new RollingPastCalendar(gmtTimeZone, Locale.ENGLISH);
  // set sate to 1970-01-01 00:00:00 GMT
  Date epoch = new Date(0);
  if(datePattern != null)
  {
    for(int i = TOP_OF_MINUTE; i <= TOP_OF_MONTH; i++)
    {
      SimpleDateFormat simpleDateFormat = new SimpleDateFormat(datePattern);
      simpleDateFormat.setTimeZone(gmtTimeZone); // do all date formatting in GMT
      String r0 = simpleDateFormat.format(epoch);
      rollingPastCalendar.setType(i);
      Date next = new Date(rollingPastCalendar.getNextCheckMillis(epoch));
      String r1 =  simpleDateFormat.format(next);

      //System.out.println("Type = "+i+", r0 = "+r0+", r1 = "+r1);
      if(r0 != null && r1 != null && !r0.equals(r1))
      {
        return i;
      }
    }
  }

  return TOP_OF_TROUBLE; // Deliberately head for trouble...
}

/**
   Rollover the current file to a new file.
*/
void rollOver() throws IOException
```

```
{
  /* Compute filename, but only if datePattern is specified */
  if (datePattern == null) {
    errorHandler.error("Missing DatePattern option in rollOver().");
    return;
  }

  String datedFilename = fileName+sdf.format(now);
  // It is too early to roll over because we are still within the
  // bounds of the current interval. Rollover will occur once the
  // next interval is reached.
  if (scheduledFilename.equals(datedFilename)) {
    return;
  }

  // close current file, and rename it to datedFilename
  this.closeFile();

  File target  = new File(scheduledFilename);
  if (target.exists()) {
    target.delete();
  }

  File file = new File(fileName);
  boolean result = file.renameTo(target);
  if(result)
  {
    LogLog.debug(fileName +" -> "+ scheduledFilename);

    // If maxBackups <= 0, then there is no file renaming to be done.
    if(maxBackupIndex > 0)
    {
      // Delete the oldest file, to keep Windows happy.
      file = new File(fileName + dateBefore());

      if (file.exists())
        file.delete();
    }
  }
  else
  {
    LogLog.error("Failed to rename [[+fileName+]] to [[+scheduledFilename+]].");
  }

  try
  {
    // This will also close the file. This is OK since multiple
    // close operations are safe.
    this.setFile(fileName, false, this.bufferedIO, this.bufferSize);
  }
  catch(IOException e) {
    errorHandler.error("setFile("+fileName+", false) call failed.");
  }
  scheduledFilename = datedFilename;
}

private String dateBefore()
{
  String dataAnte = "";


  if(datePattern != null)
  {
    SimpleDateFormat simpleDateFormat = new SimpleDateFormat(datePattern);

    dataAnte =  simpleDateFormat.format(new Date(rpc.getPastCheckMillis(new Date(), maxBackupIndex)));
  }

  return dataAnte;
}
```

```
  /**
   * This method differentiates DailyRollingFileAppender from its
   * super class.
   *
   * <p>Before actually logging, this method will check whether it is
   * time to do a rollover. If it is, it will schedule the next
   * rollover time and then rollover.
   * */
  protected void subAppend(LoggingEvent event)
  {
    long n = System.currentTimeMillis();

    if (n >= nextCheck)
    {
      now.setTime(n);
      nextCheck = rpc.getNextCheckMillis(now);

      try
      {
        rollOver();
      }
      catch(IOException ioe)
      {
        LogLog.error("rollOver() failed.", ioe);
      }
    }

    super.subAppend(event);
   }

  /*
   * DEBUG
  */
  public static void main(String args[])
  {
    DailyMaxRollingFileAppender dmrfa = new DailyMaxRollingFileAppender();

    dmrfa.setDatePattern("'.'yyyy-MM-dd-HH-mm");

    dmrfa.setFile("prova");

    System.out.println("dmrfa.getMaxBackupIndex():" + dmrfa.getMaxBackupIndex());

    dmrfa.activateOptions();

    for(int i = 0; i < 5; i++)
    {
      dmrfa.subAppend(null);

      try
      {
        Thread.sleep(60000);
      }
      catch (InterruptedException ex)
      {
      }

      System.out.println("Fine attesa");
    }
  }
}

/**
 *  RollingPastCalendar is a helper class to DailyMaxRollingFileAppender.
 *  Given a periodicity type and the current time, it computes the
 *  past maxBackupIndex date.
 * */
class RollingPastCalendar extends RollingCalendar
{
  RollingPastCalendar() {
    super();
```

```
}

RollingPastCalendar(TimeZone tz, Locale locale) {
  super(tz, locale);
}

public long getPastCheckMillis(Date now, int maxBackupIndex)
{
  return getPastDate(now, maxBackupIndex).getTime();
}

public Date getPastDate(Date now, int maxBackupIndex)
{
  this.setTime(now);

  switch(type)
  {
    case DailyRollingFileAppender.TOP_OF_MINUTE:
         this.set(Calendar.SECOND, this.get(Calendar.SECOND));
         this.set(Calendar.MILLISECOND, this.get(Calendar.MILLISECOND));
         this.set(Calendar.MINUTE, this.get(Calendar.MINUTE) - maxBackupIndex);
         break;

    case DailyRollingFileAppender.TOP_OF_HOUR:
         this.set(Calendar.MINUTE, this.get(Calendar.MINUTE));
         this.set(Calendar.SECOND, this.get(Calendar.SECOND));
         this.set(Calendar.MILLISECOND, this.get(Calendar.MILLISECOND));
         this.set(Calendar.HOUR_OF_DAY, this.get(Calendar.HOUR_OF_DAY) - maxBackupIndex);
         break;

    case DailyRollingFileAppender.HALF_DAY:
         this.set(Calendar.MINUTE, this.get(Calendar.MINUTE));
         this.set(Calendar.SECOND, this.get(Calendar.SECOND));
         this.set(Calendar.MILLISECOND, this.get(Calendar.MILLISECOND));
         int hour = get(Calendar.HOUR_OF_DAY);
         if(hour < 12)
         {
           this.set(Calendar.HOUR_OF_DAY, 12);
         }
         else
         {
           this.set(Calendar.HOUR_OF_DAY, 0);
         }
         this.set(Calendar.DAY_OF_MONTH, this.get(Calendar.DAY_OF_MONTH) - maxBackupIndex);

         break;

    case DailyRollingFileAppender.TOP_OF_DAY:
         this.set(Calendar.HOUR_OF_DAY, this.get(Calendar.HOUR_OF_DAY));
         this.set(Calendar.MINUTE, this.get(Calendar.MINUTE));
         this.set(Calendar.SECOND, this.get(Calendar.SECOND));
         this.set(Calendar.MILLISECOND, this.get(Calendar.MILLISECOND));
         this.set(Calendar.DATE, this.get(Calendar.DATE) - maxBackupIndex);
         break;

    case DailyRollingFileAppender.TOP_OF_WEEK:
         this.set(Calendar.DAY_OF_WEEK, getFirstDayOfWeek());
         this.set(Calendar.HOUR_OF_DAY, this.get(Calendar.HOUR_OF_DAY));
         this.set(Calendar.MINUTE, this.get(Calendar.MINUTE));
         this.set(Calendar.SECOND, this.get(Calendar.SECOND));
         this.set(Calendar.MILLISECOND, this.get(Calendar.MILLISECOND));
         this.set(Calendar.WEEK_OF_YEAR, this.get(Calendar.WEEK_OF_YEAR) - maxBackupIndex);
         break;

    case DailyRollingFileAppender.TOP_OF_MONTH:
         this.set(Calendar.DATE, this.get(Calendar.DATE));
         this.set(Calendar.HOUR_OF_DAY, this.get(Calendar.HOUR_OF_DAY));
         this.set(Calendar.MINUTE, this.get(Calendar.MINUTE));
         this.set(Calendar.SECOND, this.get(Calendar.SECOND));
         this.set(Calendar.MILLISECOND, this.get(Calendar.MILLISECOND));
         this.set(Calendar.MONTH, this.get(Calendar.MONTH) - maxBackupIndex);
```

```
            break;

      default:
            throw new IllegalStateException("Unknown periodicity type.");
    }

    return getTime();
  }
}
```

I'll hope can be usefull. MSC Recruitment Indonesia & MSC Crewing Indonesia | Building Signage | Anti Rayap | Pintu dan Jendela