```java
package com.pjm.tools.drhub.service;

import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

import org.apache.commons.io.FileUtils;
import org.apache.commons.io.IOUtils;
import org.apache.commons.lang3.time.FastDateFormat;
import org.joda.time.DateTime;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.pjm.esuite.cli.api.PjmRemoteCommand;
import com.pjm.esuite.cli.api.PjmResult;
import com.pjm.esuite.cli.http.PjmClient;
import com.pjm.tools.drhub.jpa.enums.EventType;
import com.pjm.web.rest.RestUtil;

import ch.qos.logback.classic.Level;
import ch.qos.logback.classic.LoggerContext;

/**
 * Poller client that retrieves unacknowledged events from DR Hub and
 * acknowledges them. This client is to be distributed to CSPs to meet tariff
 * requirements for acknowledging events.
 *
 */
public class Poller {

   /**
    * Logger for this class
    */
   private static final Logger LOG = LoggerFactory.getLogger(Poller.class);

   public static final FastDateFormat FILE_DATE_FORMAT =
FastDateFormat.getInstance("yyyyMMdd'T'HHmmss");
   public static final FastDateFormat DIRECTORY_DATE_FORMAT =
FastDateFormat.getInstance("yyyyMMdd");

   public static final String DOWNLOAD_DIRECTORY = "download";

   private static final String propertyFile = "config.properties";
   private static final ScheduledExecutorService scheduler =
Executors.newScheduledThreadPool(1);

   /**
```

```
    * Main entry point for Poller. This method sets up a scheduled
    * {@link Runnable} that uses the PJM CLI API to retrieve and acknowledge
all
    * unacknowledged events on a minute interval set in the configuration.
    *
    * @param args the command line arguments
    */
   public static void main(final String[] args) {
      LOG.info("Starting process");

      LOG.info("Getting properties file");
      final Properties properties = new Properties();
      try (InputStream propertyFileStream = new
FileInputStream(propertyFile)) {
         properties.load(propertyFileStream);
      } catch (final IOException exception) {
         LOG.error(
                  "Error: unable to locate properties file.  Please make sure
your properties file is located in the assembly folder.",
                  exception);
      }

      // Get Properties
      LOG.info("Getting properties from /assembly/config.properties");
      final String username = properties.getProperty("username");
      final String password = properties.getProperty("password");
      final String url = properties.getProperty("url");
      final String logging = properties.getProperty("logging");
      final String intervalMinsStr = properties.getProperty("intervalmins");
      final Long intervalMins = Long.parseLong(intervalMinsStr);
      final Integer retainDays =
Integer.parseInt(properties.getProperty("retaindays"));
      final List<EventType> eventTypes = getEventTypes(properties);

      setLogLevel(logging);

      // Create Runnable method
      final Runnable downloadEvents = new Runnable() {
         @Override
         public void run() {
            final DateTime now = DateTime.now();
            for (final EventType eventType : eventTypes) {
               downloadEvents(eventType, username, password, url, logging);
            }
            LOG.info("Next execution at {}",
now.plusMinutes(intervalMins.intValue()));
         }
      };

      final Runnable purgeDownloadsPastThreshold = new Runnable() {
         @Override
         public void run() {
            purgeDownloadsPastThreshold(DateTime.now(), retainDays);
         }
      };

      // Call DR Hub
```

```
      scheduler.scheduleAtFixedRate(downloadEvents, 0L, intervalMins,
TimeUnit.MINUTES);
      scheduler.scheduleAtFixedRate(purgeDownloadsPastThreshold, 0L, 1l,
TimeUnit.DAYS);

   }

   /**
    * Downloads unacknowledged events and acknowledges them.
    *
    * @param eventType - the {@link EventType} to process
    * @param username - the username used for webservice interaction
    * @param password - the password user for webservice interaction
    * @param url - the base URL of the webservices
    * @param logging - the log level used when processing events.
    */
   @SuppressWarnings("unchecked")
   static void downloadEvents(final EventType eventType, final String
username, final String password, final String url,
            final String logging) {
      OutputStream outputStream = null;
      try {

         // Download Events Command
         final PjmRemoteCommand getEventCommand = new PjmRemoteCommand();
         getEventCommand.setServiceUrl(url);
         getEventCommand.setUsername(username);
         getEventCommand.setPassword(password);
         getEventCommand.setLogLevel(logging);
         getEventCommand.setCsvToXml(false);

         // Acknowledge Events Command
         final PjmRemoteCommand acknowledgeCommand = new PjmRemoteCommand();
         acknowledgeCommand.setServiceUrl(url);
         acknowledgeCommand.setUsername(username);
         acknowledgeCommand.setPassword(password);
         acknowledgeCommand.setLogLevel(logging);
         acknowledgeCommand.setCsvToXml(false);
         acknowledgeCommand.setTimeout(180000);

         switch (eventType) {
         case LOAD_MANAGEMENT_EVENT:

getEventCommand.setAction("rest/secure/download/event/loadmanagement/acknowle
dgeable");

acknowledgeCommand.setAction("rest/secure/upload/event/loadmanagement/acknowl
edge/");
            break;
         case REAL_TIME_DISPATCH_EVENT:

getEventCommand.setAction("rest/secure/download/event/realtimedispatch/acknow
ledgeable");

acknowledgeCommand.setAction("rest/secure/upload/event/realtimedispatch/ackno
wledge/");
            break;
```

```
         case SYNCHRONIZED_RESERVE_EVENT:

getEventCommand.setAction("rest/secure/download/event/synchronizedreserve/ack
nowledgeable");

acknowledgeCommand.setAction("rest/secure/upload/event/synchronizedreserve/ac
knowledge/");
            break;
         case ZONAL_EMERGENCY_ENERGY_EVENT:

getEventCommand.setAction("rest/secure/download/event/zonalemergencyenergy/ac
knowledgeable");

acknowledgeCommand.setAction("rest/secure/upload/event/zonalemergencyenergy/a
cknowledge");
            break;
         case DAY_AHEAD_ENERGY_EVENT:
         case LOAD_MANAGEMENT_SUMMARY:
         case REAL_TIME_DISPATCH_SUMMARY:
         case ZONAL_EMERGENCY_ENERGY_SUMMARY:
         case LOAD_MANAGEMENT_TEST:
         case LOAD_MANAGEMENT_RETEST:
         default:
            throw new UnsupportedOperationException("Unknown or unsupported
event type");
         }

         final DateTime now = DateTime.now();
         final File downloadDirectory = getCurrentDirectory(now);
         final File unacknowledgedEventResponse =
getFileForDownload(eventType, "UNACKNOWLEDGED", now,
                  downloadDirectory);

         outputStream = new BufferedOutputStream(new
FileOutputStream(unacknowledgedEventResponse));
         getEventCommand.setOutputStream(outputStream);

         // Download Results
         LOG.info("Attempting to download unacknowleged events for event type
{}", eventType);
         final PjmResult downloadResult = PjmClient.execute(getEventCommand);
         if (downloadResult.getResultCode() == PjmResult.CLI_SUCCESS) {
            LOG.info("Download succeeded for event type {} stored to {}",
eventType,
                     unacknowledgedEventResponse.getAbsolutePath());
         } else {
            LOG.error("Download failed for event type {}", eventType);
         }

         // Handle unacknowledged events

acknowledgeCommand.setUploadFileName(unacknowledgedEventResponse.getName());
         acknowledgeCommand.setInputStream(new
FileInputStream(unacknowledgedEventResponse));

         final File acknowledgedEventResponse = getFileForDownload(eventType,
"ACKNOWLEDGED", now, downloadDirectory);
```

```
         outputStream = new BufferedOutputStream(new
FileOutputStream(acknowledgedEventResponse));
         acknowledgeCommand.setOutputStream(outputStream);

         LOG.info("Attempting to acknowleged event ids for event type {}",
eventType);
         final PjmResult uploadResult =
PjmClient.execute(acknowledgeCommand);
         if (uploadResult.getResultCode() == PjmResult.CLI_SUCCESS) {
            LOG.info("Acknowledge successed.  Download acknowledgement
results for event type {} stored to {}",
                     eventType, acknowledgedEventResponse.getAbsolutePath());
         } else {
            LOG.error("Acknowleged failed for ids for Event {} ", eventType);
         }

      } catch (final Exception ex) {
         LOG.error("Unexpected exception occured", ex);
      } finally {
         IOUtils.closeQuietly(outputStream);
      }
   }

   /**
    * Builds {@link List} of {@link EventType EventTypes} to retrieve.
    *
    * @param properties {@link Properties}.
    * @return {@link List} of {@link EventType EventTypes}.
    */
   static List<EventType> getEventTypes(final Properties properties) {
      final List<EventType> eventTypes = new ArrayList<>();
      if
(Boolean.parseBoolean(properties.getProperty("LOAD_MANAGEMENT_EVENT"))) {
         eventTypes.add(EventType.LOAD_MANAGEMENT_EVENT);
      }
      if
(Boolean.parseBoolean(properties.getProperty("REAL_TIME_DISPATCH_EVENT"))) {
         eventTypes.add(EventType.REAL_TIME_DISPATCH_EVENT);
      }
      if
(Boolean.parseBoolean(properties.getProperty("SYNCHRONIZED_RESERVED_EVENT")))
{
         eventTypes.add(EventType.SYNCHRONIZED_RESERVE_EVENT);
      }
      if
(Boolean.parseBoolean(properties.getProperty("ZONAL_EMERGENCY_ENERGY_EVENT"))
) {
         eventTypes.add(EventType.ZONAL_EMERGENCY_ENERGY_EVENT);
      }
      return eventTypes;
   }

   /**
    * Creates and returns a {@link File} whose name consists of the given
    * {@link EventType} and {@link DateTime} separated by a "-" with the xml
    * extension.
    *
```

```
     * @param eventType - the {@link EventType} used in the file name
     * @param type - the type of file, UNACKNOWLEDGED or ACKNOWLEDGED
     * @param date - the {@link DateTime} used in the file name
     * @param downloadDirectory - the directory in which to create the file
     * @return a {@link File} that represents the newly created file
     * @throws IOException if the file creation is unsuccessful
     */
    static File getFileForDownload(final EventType eventType, final String
type, final DateTime date,
                final File downloadDirectory) throws IOException {
        final String filename = new
StringBuilder().append(eventType.toString()).append("-
").append(type).append("-")

.append(FILE_DATE_FORMAT.format(date.toDate())).append(".").append(RestUtil.E
XTENSION_XML).toString();
        final File file = new File(downloadDirectory, filename);
        file.createNewFile();
        return file;
    }

    /**
     * Returns a {@link File} object that refers to the download directory
named
     * by the date parameter.
     *
     * @param date - the {@link DateTime} used when creating the directory
     * @return a {@link File} that represents a directory
     */
    static File getCurrentDirectory(final DateTime date) {
        final File downloadDirectory = new File(DOWNLOAD_DIRECTORY);
        final File currentDirectory = new File(downloadDirectory,
DIRECTORY_DATE_FORMAT.format(date.toDate()));
        if (!currentDirectory.exists() && !currentDirectory.mkdir()) {
            throw new RuntimeException("Could not create directory for
downloads");
        }
        return currentDirectory;
    }

    /**
     * Deletes directories in the download directory that are older than the
     * retention threshold. This threshold is computed as beginning of now
minus
     * retainDays days.
     *
     * @param now - the current date to work from
     * @param retainDays - the number of days to retain directories
     */
    static void purgeDownloadsPastThreshold(final DateTime now, final Integer
retainDays) {
        LOG.info("Purging directories for date: {}, with retainDays: {}", now,
retainDays);
        final File downloadDirectory = new File(DOWNLOAD_DIRECTORY);
        final DateTime thresholdDate = now.withTime(0, 0, 0,
0).minusDays(retainDays);
        for (final File file : downloadDirectory.listFiles()) {
```

```
         try {
            LOG.info("Processing file: {}", file.getAbsolutePath());
            if (!file.isDirectory()) {
               continue;
            }
            final String directoryName = file.getName();
            final DateTime directoryDate = new
DateTime(DIRECTORY_DATE_FORMAT.parse(directoryName));
            if (directoryDate.isBefore(thresholdDate)) {
               LOG.info("Purging directory: {}", file.getAbsolutePath());
               FileUtils.deleteDirectory(file);
            }
         } catch (final Exception e) {
            LOG.debug("Unexpected exception occured", e);
         }
      }
   }

   /**
    * Sets the logback log level for com.pjm based on the param string.
    *
    * @param logging - the logging level (TRACE, DEBUG, INFO, WARN, ERROR)
    */
   private static void setLogLevel(final String logging) {
      final LoggerContext loggerContext = (LoggerContext)
LoggerFactory.getILoggerFactory();
      final ch.qos.logback.classic.Logger rootLogger =
loggerContext.getLogger("com.pjm");
      rootLogger.setLevel(Level.valueOf(logging));
   }

}
```