C:\checkout\openii3\project\java\source\org\openeai\implementations\services\egs\com
May 9, 2006 10:01:36 AM

```
 1  /*****************************************************************************
 2   $Source: /cvs/repositories/openii3/project/java/source/org/openeai/implementat
 3   $Revision: 1.1 $
 4  *****************************************************************************
 5
 6  /*********************************************************************
 7  This file is part of the OpenEAI sample, reference implementation,
 8  and deployment management suite created by Tod Jackson
 9  (tod@openeai.org) and Steve Wheat (steve@openeai.org).
10
11  Copyright (C) 2003 The OpenEAI Software Foundation
12
13  This program is free software; you can redistribute it and/or modify
14  it under the terms of the GNU General Public License as published by
15  the Free Software Foundation; either version 2 of the License, or
16  (at your option) any later version.
17
18  This program is distributed in the hope that it will be useful,
19  but WITHOUT ANY WARRANTY; without even the implied warranty of
20  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
21  GNU General Public License for more details.
22
23  You should have received a copy of the GNU General Public License
24  along with this program; if not, write to the Free Software
25  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
26
27  For specific licensing details and examples of how this software
28  can be used to implement integrations for your enterprise, visit
29  http://www.OpenEai.org/licensing.
30  */
31
32  package org.openeai.implementations.services.egs.commands;
33
34  // Core Java and utilities
35  import java.io.*;
36  import java.util.*;
37  import javax.jms.*;
38
39  // Log4j
40  import org.apache.log4j.*;
41
42  // General OpenEAI Foundation
43  import org.openeai.afa.*;
44  import org.openeai.config.*;
45  import org.openeai.dbpool.*;
46  import org.openeai.jms.producer.*;
47  import org.openeai.threadpool.*;
48  import org.openeai.moa.*;
49
50  // OpenEAI Implementations Message Objects
51  import org.any_openeai_enterprise.moa.objects.resources.v1_0.Greetee;
52  import org.any_openeai_enterprise.moa.jmsobjects.coreapplication.v1_0.Greeting;
53
54  /**
55   * This is an OpenEAI implementation of the classic "Hello, World" example.
56   * This command demonstrates the example service called the
57   * EnterpriseGreetingService by reading a file with the names of people or
```

```
 58   * things to greet and using those names to send requests to the
 59   * EnterpriseGreetingService to generate greetings for these names.
 60   * <P>
 61   * 1. org.any-openeai-enterprise.CoreApplication.Greeting.Generate-Request
 62   * <P>
 63   * Specifically, this command reads an input file specified in the deployment
 64   * descriptor. The format of the expected input file is one name of a person
 65   * or thing to be greeted per line of the input file. The command uses the
 66   * names in the file as the FullName for Greetee objects required to build
 67   * the Greeting.Generate-Request.  The command logs runtime events and
 68   * maintains some basic runtime statistics for demonstration purposes.
 69   * <P>
 70   * <TABLE BORDER=2 CELLPADDING=5 CELLSPACING=2>
 71   * <TR>
 72   * <TH>Name</TH>
 73   * <TH>Required</TH>
 74   * <TH>Description</TH>
 75   * </TR>
 76   * <TR HALIGN="left" VALIGN="top">
 77   * <TD>inputFileName</TD>
 78   * <TD>yes</TD>
 79   * <TD>The path and name of the input file to read containing names of the
 80   * people or things to greet.</TD>
 81   * </TR>
 82   * <TR HALIGN="left" VALIGN="top">
 83   * <TD>useThreads</TD>
 84   * <TD>no (default is 'false')</TD>
 85   * <TD>Send the generate requests in a single thread or use a thread pool
 86   * as configured in the deployment descriptor. The default value is false,
 87   * indicating not to use threads, if this property is not specified in the
 88   * deployment descriptor.</TD>
 89   * </TR>
 90   * </TABLE>
 91   * <P>
 92   * @author      Steve Wheat (steve@openeai.org)
 93   * @version     1.0 - 27 September 2003
 94   */
 95  public class GreetingGenerationCommand extends ScheduledCommandImpl implements
 96    ScheduledCommand {
 97
 98    private boolean m_useThreads = false;
 99    private ProducerPool m_egsProducerPool = null;
100    private ThreadPool m_threadPool = null;
101    private int m_successCount = 0;
102    private int m_errorCount = 0;
103    private String m_inputFileName = null;
104    private int m_sleepInterval = 500;
105    private Date m_applicationStartTime = null;
106
107    /**
108     * @param CommandConfig
109     * @throws InstantiationException
110     * <P>
111
112     * This constructor initializes the command using a CommandConfig object. It
113     * invokes the constructor of the ancestor, ScheduledCommandImpl, and then
114     * gets the value of the application start time, gets and sets genernal
115     * properties for the command, gets and sets the value of the inputFileName,
116     * and gets and sets the value of the useThreads indicator. Note that if
117     * the useThreads property is not specified in the configuration,
```

```
118     * its default value is false.
119     */
120    public GreetingGenerationCommand(CommandConfig cConfig) throws
121      InstantiationException {
122      super(cConfig);
123      logger.info("[GreetingGenerationCommand] Initializing...");
124
125      // Get the application start time.
126      setApplicationStartTime(getCurrentTime());
127
128      // Set the properties for this command.
129      try {
130        PropertyConfig pConfig = (PropertyConfig)getAppConfig()
131          .getObject("GreetingGenerationCommandProperties");
132        setProperties(pConfig.getProperties());
133      }
134      catch (EnterpriseConfigurationObjectException ecoe) {
135        // An error occurred retrieving a property config from AppConfig. Log it
136        // and throw an exception.
137        String errMsg = "An error occurred retrieving a property config from " +
138          "AppConfig. The exception is: " + ecoe.getMessage();
139        logger.fatal("[GreetingGenerationCommand] " + errMsg);
140        throw new InstantiationException(errMsg);
141      }
142
143      // Get the value of the useThreads property, so we can determine whether
144      // or not to use threads.
145      if (getProperties().getProperty("useThreads", "false")
146        .equalsIgnoreCase("true")) {
147
148        setUseThreads(true);
149      }
150      logger.info("[GreetingGenerationCommand] The value of useThreads is: " +
151        getUseThreads());
152
153      // Get the value of the inputFileName property.
154      String inputFileName = getProperties().getProperty("inputFileName");
155      if (inputFileName == null) {
156        String errMsg = "Missing 'inputFileName' property in the " +
157          "deployment descriptor. Can't continue.";
158        logger.fatal("[GreetingGenerationCommand] " + errMsg);
159        throw new InstantiationException(errMsg);
160      }
161      setInputFileName(inputFileName);
162      logger.info("[GreetingGenerationCommand] Using input file located at: " +
163        getInputFileName());
164
165      // If we are going to use threads, get a thread pool from AppConfig to use.
166      if (getUseThreads() == true) {
167        try {
168          ThreadPool threadPool = null;
169          threadPool = (ThreadPool)getAppConfig()
170            .getObject("GreetingGenerationThreadPool");
171          setThreadPool(threadPool);
172        }
173        catch (EnterpriseConfigurationObjectException ecoe) {
174          // An error occurred retrieving a thread pool from AppConfig. Log it an
175          // throw an exception.
176          String errMsg = "An error occurred retrieving a thread pool from " +
177            "AppConfig. The exception is: " + ecoe.getMessage();
```

```
178           logger.fatal("[GreetingGenerationCommand] " + errMsg);
179           throw new InstantiationException(errMsg);
180         }
181       }
182
183       // Get the producer pool we specified in the deployment descriptor from
184       // AppConfig. There should be one producer pool for messaging with
185       // the EnterpriseGreetingService named P2pEgsProducer.
186       try {
187         // Get the producer pool to use for communicating with EGS.
188         ProducerPool egsProducerPool = (ProducerPool)getAppConfig()
189           .getObject("P2pEgsProducer");
190         setEgsProducerPool(egsProducerPool);
191       }
192       catch(EnterpriseConfigurationObjectException ecoe) {
193         // An error occurred retrieving a required producer pool from AppConfig.
194         // Log it and throw an exception.
195         String errMsg = "An error occurred retrieving a producer pool from " +
196           "AppConfig. The exception is: " + ecoe.getMessage();
197         logger.fatal("[GreetingGenerationCommand] " + errMsg);
198         throw new InstantiationException(errMsg);
199       }
200
201       logger.info("[GreetingGenerationCommand] Initialization complete.");
202     }
203
204     /**
205      * @return int, a return code for the command execution
206      * @throws ScheduledCommandException
207      * <P>
208      * This execute method gets the start time for this execution and resets
209      * execution specific counters, which are used in the execution summary
210      * statistics. It then attempts to read the input file specified into a list.
211      * If the input file does not exist or if there is an error reading the input
212      * file, the command throws a ScheduledCommandException with details of the
213      * error.
214      * <P>
215      * If the file has been read into memory successfully, the command pulls the
216      * file lines out of the list for processing ignoring any blank lines. The
217      * command gets a point-to-point producer for sending requests to use in
218      * processing each line. If the command is supposed to use threads, it
219      * instantiates a ProcessInputLine transaction for each line and adds them to
220      * the thread pool. In adding jobs to the thread pool, the command must
221      * attempt to add each job until it is successfully added to the pool. If the
222      * thread pool is throttled to check before processing, it is possible that
223      * jobs cannot be accepted by the pool at any given point in time. If the
224      * command is not supposed to use threads, it instatiates a ProcessInputLine
225      * transaction for each line and executes them in sequence in the main
226      * execution thread of the command. If threads were used to process the input
227      * file, the command waits for all threads to complete, then calculates and
228      * outputs summary statistics for the current execution, and then returns.
229      */
230     public int execute() throws ScheduledCommandException {
231       logger.info("[GreetingGenerationCommand] Executing...");
232
233       // Get the start time for this execution.
234       java.util.Date executionStartTime = getCurrentTime();
235
236       // Reset counters for this execution.
237       resetCounters();
```

```
238
239       // Read the input file into a vector.
240       logger.info("[GreetingGenerationCommand.execute] Reading file " +
241         getInputFileName());
242
243       String inputFileLine;
244       Reader reader = null;
245
246       try { reader = new FileReader(getInputFileName()); }
247       catch (FileNotFoundException fnfe) {
248         // The input file does not exist. Log it and throw an exception.
249         String errMsg = "The input file " + getInputFileName() + " does not " +
250           "exist.";
251         logger.fatal("[GreetingGenerationCommand.exectue] " + errMsg);
252         throw new ScheduledCommandException(errMsg);
253       }
254
255       BufferedReader br = new BufferedReader(reader);
256       Vector vLines = new Vector();
257       logger.debug("[GreetingGenerationCommand.execute] Started reading the " +
258         "input file into memory.");
259       try {
260         while (br.ready()) {
261           inputFileLine = br.readLine();
262           if (inputFileLine != null && inputFileLine.trim().length() > 0) {
263             vLines.add(inputFileLine);
264           }
265         }
266       }
267       catch (IOException ioe) {
268         // An error occurred reading the input file. Log it and throw an
269         // exception.
270         String errMsg = "An error occurred reading the input file. The " +
271           "exception is: " + ioe.getMessage();
272         logger.fatal("[GreetingGenerationCommand.exectue] " + errMsg);
273         throw new ScheduledCommandException(errMsg);
274       }
275       logger.debug("[GreetingGenerationCommand.execute] Finished reading the " +
276         "input file into memory.");
277       logger.info("[GreetingGenerationCommand.execute] There are " + vLines.size(
278         + " lines in the input file.");
279
280       // Pull the inputFileLines back out of the vector for processing.
281       int numberOfLines = vLines.size();
282       for (int lineNumber=1; lineNumber <= numberOfLines; ++lineNumber) {
283         String currentLine = (String)vLines.elementAt(0);
284         vLines.removeElementAt(0);
285
286         // Skip any blank lines.
287         if (currentLine == null || currentLine.length() == 0) {
288           logger.debug("[GreetingGenerationCommand.execute] Skipping line " +
289             lineNumber + ", because it is blank.");
290           continue;
291         }
292
293         // Get a producer from the producer pool to use in processing this line.
294         PointToPointProducer p2p = null;
295         try {
296           p2p = (PointToPointProducer)getEgsProducerPool().getProducer();
297         }
```

```
298         catch (JMSException jmse) {
299           // An error occurred getting a producer from the producer pool. Log it
300           // and throw an exception.
301           String errMsg = "An error occurred getting a producer from the " +
302             "producer pool. The exception is: " + jmse.getMessage();
303           logger.fatal("[GreetingGenerationCommand.execute] " + errMsg);
304           throw new ScheduledCommandException(errMsg);
305         }
306
307         // If we are using threads, the delegate the processing of this input lin
308         // to the threadpool.
309         if (getUseThreads()) {
310           // If this thread pool is set to check for available threads before
311           // adding jobs to the pool, it may throw an exception indicating it
312           // is busy when we try to add a job. We need to catch that exception
313           // and try to add the job until we are successful.
314           boolean jobAdded = false;
315           while (jobAdded == false) {
316             try {
317               getThreadPool().addJob(new ProcessInputLine(lineNumber, currentLine
318                 p2p));
319               jobAdded = true;
320             }
321             catch (ThreadPoolException tpe) {
322               // The thread pool is busy. Log it and sleep briefly to try to add
323               // the job again later.
324               String msg = "The thread pool is busy. Sleeping for " +
325                 getSleepInterval() + " milliseconds.";
326               logger.debug("[GreetingGenerationCommand.execute] " + msg);
327               try { Thread.sleep(getSleepInterval()); }
328               catch (InterruptedException ie) {
329                 // An error occurred while sleeping to allow threads in the pool
330                 // to clear for processing. Log it and throw and exception.
331                 String errMsg = "An error occurred while sleeping to allow " +
332                   "threads in the pool to clear for processing. The exception " +
333                   "is " + ie.getMessage();
334                 logger.fatal("[GreetingGenerationCommand.execute] " + errMsg);
335                 throw new ScheduledCommandException(errMsg);
336               }
337             }
338           }
339         }
340         // Otherwise, if we are not using threads, just process the current line
341         // in the current command execution thread.
342         else {
343           new ProcessInputLine(lineNumber, currentLine, p2p).run();
344         }
345       }
346
347       // If we are using threads, wait for all threads to complete.
348       if (getUseThreads()) {
349         while (getThreadPool().getJobsInProgress() > 0) {
350           try { Thread.sleep(getSleepInterval()); }
351           catch (InterruptedException ie) {
352             // An error occurred while sleeping to allow threads in the pool
353             // to complete. Log it and throw and exception.
354             String errMsg = "An error occurred while sleeping to allow " +
355               "threads in the pool to complete. The exception " +
356               "is " + ie.getMessage();
357             logger.fatal("[GreetingGenerationCommand.execute] " + errMsg);
```

```
358             throw new ScheduledCommandException(errMsg);
359           }
360         }
361       }
362
363       logger.info("[GreetingGenerationCommand.execute] All lines in the input " +
364         "file have been processed, summarizing...");
365
366       // Get the end time and summarize run-time statistics.
367       java.util.Date executionEndTime = getCurrentTime();
368       long executionTime =
369         ((executionEndTime.getTime() - executionStartTime.getTime()) / 1000);
370       long totalTime =
371         ((executionEndTime.getTime() - getApplicationStartTime().getTime()) /
372         1000);
373       logger.info("[GreetingGenerationCommand] Application start time: " +
374         getApplicationStartTime());
375       logger.info("[GreetingGenerationCommand]   Execution start time: " +
376         executionStartTime);
377       logger.info("[GreetingGenerationCommand]     Execution end time: " +
378         executionEndTime);
379       logger.info("[GreetingGenerationCommand]         Execution time: " +
380         executionTime + " seconds");
381       logger.info("[GreetingGenerationCommand]   Application run time: " +
382         totalTime + " seconds");
383       logger.info("[GreetingGenerationCommand]     Greeting successes: " +
384         getSuccessCount());
385       logger.info("[GreetingGenerationCommand]      Greeting failures: " +
386         getErrorCount());
387       logger.info("[GreetingGenerationCommand] Exiting.");
388
389       return 0;
390     }
391
392     /**
393      * @return java.util.Date, the current time as a date.
394      * <P>
395      * Returns the current time as a date.
396      */
397     private final java.util.Date getCurrentTime() {
398       Calendar myCal = new GregorianCalendar();
399       return myCal.getTime();
400     }
401
402     /**
403      * @param boolean, indicates whether or not to use threads.
404      * <P>
405      * Sets the use threads indicator.
406      */
407     private void setUseThreads(boolean useThreads) {
408       m_useThreads = useThreads;
409     }
410
411     /**
412      * @return boolean, indicates whether or not to use threads.
413      * <P>
414      * Returns the use threads indicator.
415      */
416     private boolean getUseThreads() {
417       return m_useThreads;
```

```
418     }
419
420     /**
421      * @param int, the sleep interval or number of milliseconds to sleep for
422      * thread operations such as waiting to add jobs or waiting for jobs
423      * to complete.
424      * <P>
425      * Sets the sleep interval.
426      */
427     private void setSleepInterval(int sleepInterval) {
428       m_sleepInterval = sleepInterval;
429     }
430
431     /**
432      * @return int, the sleep interval or number of milliseconds to sleep for
433      * thread operations such as waiting to add jobs or waiting for jobs
434      * to complete.
435      * <P>
436      * Returns the sleep interval.
437      */
438     private int getSleepInterval() {
439       return m_sleepInterval;
440     }
441
442     /**
443      * @param String, the input file name.
444      * <P>
445      * Sets the input file name.
446      */
447     private void setInputFileName(String inputFileName) {
448       m_inputFileName = inputFileName;
449     }
450
451     /**
452      * @return String, the input file name.
453      * <P>
454      * Returns the input file name.
455      */
456     private String getInputFileName() {
457       return m_inputFileName;
458     }
459
460     /**
461      * @param ThreadPool, the thread pool for the command to use.
462      * <P>
463      * Sets the thread pool for the command to use.
464      */
465     private void setThreadPool(ThreadPool threadPool) {
466       m_threadPool = threadPool;
467     }
468
469     /**
470      * @return ThreadPool, the thread pool for the command to use.
471      * <P>
472      * Returns the thread pool for the command to use.
473      */
474     private ThreadPool getThreadPool() {
475       return m_threadPool;
476     }
477
```

```
478     /**
479      * @param ProducerPool, the producer pool for the command to use to
480      * communicate with EGS.
481      * <P>
482      * Sets the producer pool for the command to use to communicate with EGS.
483      */
484     private void setEgsProducerPool(ProducerPool producerPool) {
485       m_egsProducerPool = producerPool;
486     }
487
488     /**
489      * @return ProducerPool, the producer pool for the command to use to
490      * communicate with EGS.
491      * <P>
492      * Returns the producer pool for the command to use to communicate with EGS.
493      */
494     private ProducerPool getEgsProducerPool() {
495       return m_egsProducerPool;
496     }
497
498     /**
499      * @param Date, the application start time.
500      * <P>
501      * Sets the application start time.
502      */
503     private void setApplicationStartTime(java.util.Date applicationStartTime) {
504       m_applicationStartTime = applicationStartTime;
505     }
506
507     /**
508      * @return Date, the application start time.
509      * <P>
510      * Returns the application start time.
511      */
512     private Date getApplicationStartTime() {
513       return m_applicationStartTime;
514     }
515
516     /**
517      * Resets the error count for a command execution to zero.
518      */
519     private synchronized void resetCounters() {
520       m_errorCount = 0;
521       m_successCount = 0;
522     }
523
524     /**
525      * @return int, the error count for a command execution.
526      * <P>
527      * Returns the error count for a command execution.
528      */
529     private int getErrorCount() {
530       return m_errorCount;
531     }
532
533     /**
534      * @return int, the error count for a command execution.
535      * <P>
536      * Increments the error count for a command execution.
537      */
```

```
538     private synchronized int incrementErrorCount() {
539       return ++m_errorCount;
540     }
541
542     /**
543      * @return int, the success count for a command execution.
544      * <P>
545      * Returns the success count for a command execution.
546      */
547     private int getSuccessCount() {
548       return m_successCount;
549     }
550
551     /**
552      * @return int, the success count for a command execution.
553      * <P>
554      * Increments the success count for a command execution.
555      */
556     private synchronized int incrementSuccessCount() {
557       return ++m_successCount;
558     }
559
560     /**
561      * A runnable class for processing input file lines.
562      */
563     private class ProcessInputLine implements java.lang.Runnable {
564
565       private int m_lineNumber = 0;
566       private String m_line = null;
567       private PointToPointProducer m_p2p = null;
568
569       /**
570        * @param int, the line number from the input file to processed.
571        * @param String, the line from the input file to process.
572        * @param PointToPointProducer, producer to use to send generate requests.
573        * <P>
574        * This constructor sets the values of the line number, line, and point-
575        * to-point producer fields that will be used when this transaction runs.
576        */
577       public ProcessInputLine(int lineNumber, String line, PointToPointProducer
578         p2p) {
579         m_lineNumber = lineNumber;
580         m_line = line;
581         m_p2p = p2p;
582       }
583
584       /**
585        * This method processes the input file line by getting a configured
586        * greetee object and greeting object from AppConfig. If an error occurs
587        * getting these object from AppConfig, it logs the error, increments the
588        * error counter, and returns. This sample application does not attempt to
589        * track and reattempt failed lines like any production application should.
590        * Then this method sets the value of the greetee's full name to be the
591        * trimmed value of the input line. If an error occurs setting this value,
592        * the it logs the error, increments the error counter, and returns.
593        * Then the method sends a Greeting.Generate-Request to generate a
594        * greeting using the greetee. If an error occurs generating the greeting,
595        * it logs the error, increments the error counter, and returns. Finally,
596        * this method logs the greeting that was generated for the input file line
597        * increments the success counter and returns.
```

```
598       */
599      public void run() {
600        logger.debug("[ProcessInputLine] Processing line " + m_lineNumber + ": "
601          m_line);
602
603        // Get a greeting and a greetee object from AppConfig to use in
604        // processing this line.
605        Greeting greeting = null;
606        Greetee greetee = null;
607        try {
608          greeting = (Greeting)getAppConfig().getObject("Greeting");
609          greetee = (Greetee)getAppConfig().getObject("Greetee");
610        }
611        catch (EnterpriseConfigurationObjectException ecoe) {
612          // An error occurred retrieving an object from AppConfig. Log it,
613          // increment the error count, and return.
614          String errMsg = "An error occurred retrieving an object from " +
615            "AppConfig. The exception is: " + ecoe.getMessage();
616          logger.fatal("[ProcessInputLine.run] An error occurred processing " +
617            "line number " + m_lineNumber + ": " + errMsg);
618          incrementErrorCount();
619          return;
620        }
621
622        // Set the full name of the greetee.
623        try { greetee.setFullName(m_line.trim()); }
624        catch (EnterpriseFieldException efe) {
625          // An error occurred setting the value of the full name for the
626          // greetee. Log it, increment the error count, and return.
627          String errMsg = "An error occurred setting the value of the full name "
628            + "of the greetee. The exception is: " + efe.getMessage();
629          logger.fatal("[ProcessInputLine.run] An error occurred processing " +
630            "line number " + m_lineNumber + ": " + errMsg);
631          incrementErrorCount();
632          return;
633        }
634
635        // Generate the greeting.
636        List results = null;
637        try {
638          logger.info("[ProcessInputLine.run] Sending a Greeting.Generate-" +
639            "Request for: " + greetee.getFullName());
640          results = greeting.generate(greetee, m_p2p);
641        }
642        catch (EnterpriseObjectGenerateException eoge) {
643          if (eoge.getMessage().toLowerCase().indexOf("time") != -1) {
644            // The error was a timeout error. Log it.
645            String errMsg = "Timed out waiting for reply.";
646            logger.fatal("[ProcessInputLine.run] " + errMsg);
647          }
648          else {
649            // An error occurred generating the greeting. Log it.
650            String errMsg = "An error occurred generating the greeting. The " +
651              "exception is: " + eoge.getMessage();
652            logger.fatal("[ProcessInputLine.run] " + errMsg);
653            eoge.printStackTrace();
654          }
655          // Increment the error count and return.
656          incrementErrorCount();
657          return;
```

```
658         }
659
660         // Display the greeting that was generated by EGS.
661         for (int i=0; i < results.size(); i++) {
662           greeting = (Greeting)results.get(i);
663           logger.info("[ProcessInputLine.run] Greeting returned for '" +
664             greetee.getFullName() + "' is: " + greeting.getText());
665         }
666
667         logger.debug("[ProcessInputLine.run] Completed processing line " +
668           m_lineNumber + ": " + m_line);
669
670         // Increment the success counter and return.
671         incrementSuccessCount();
672         return;
673       }
674     }
675   }
676
```