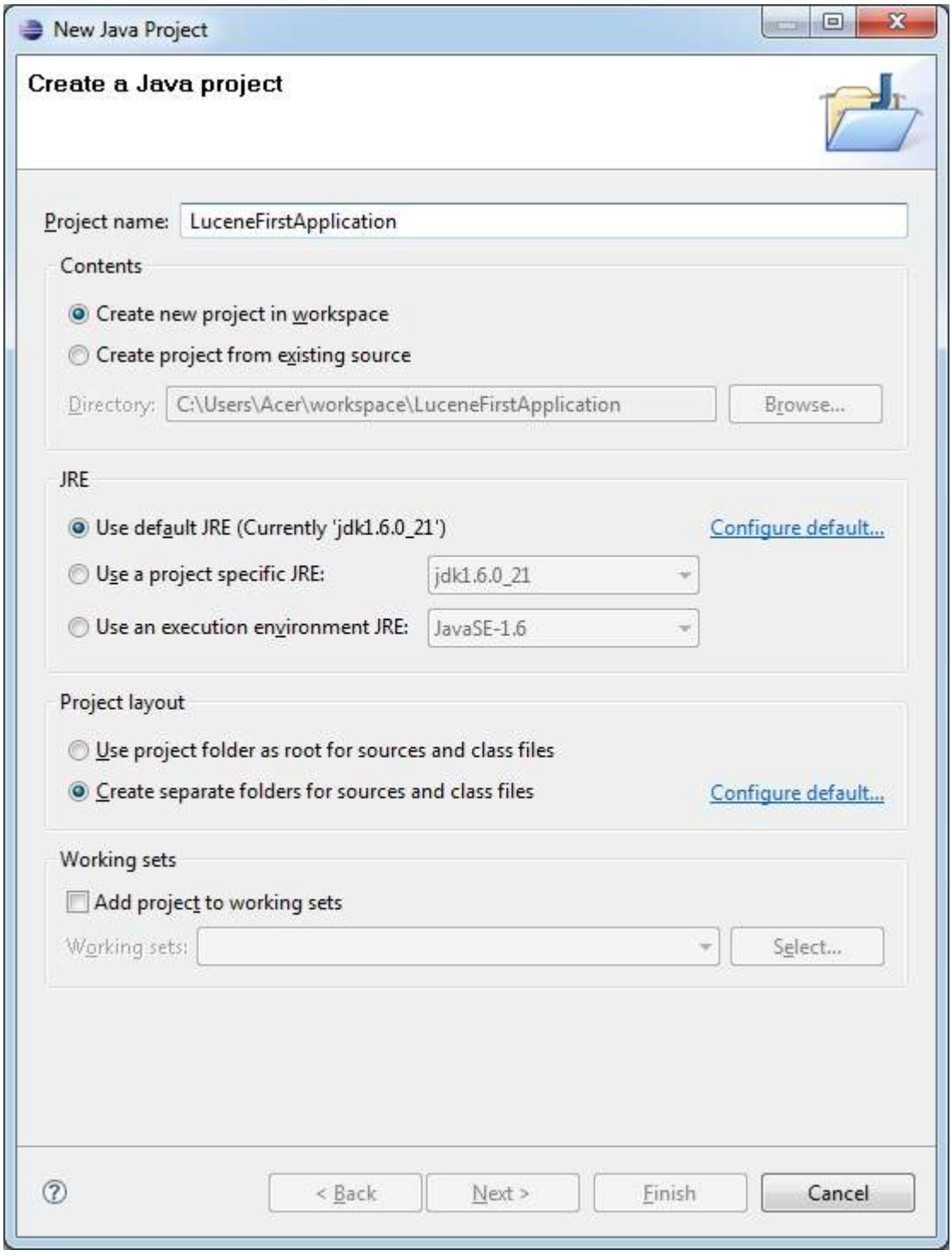
LUCENE - FIRST APPLICATION

Let us start actual programming with Lucene Framework. Before you start writing your first example using Lucene framework, you have to make sure that you have setup your Lucene environment properly as explained in Lucene - Environment Setup tutorial. I also assume that you have a little bit working knowledge with Eclipse IDE.

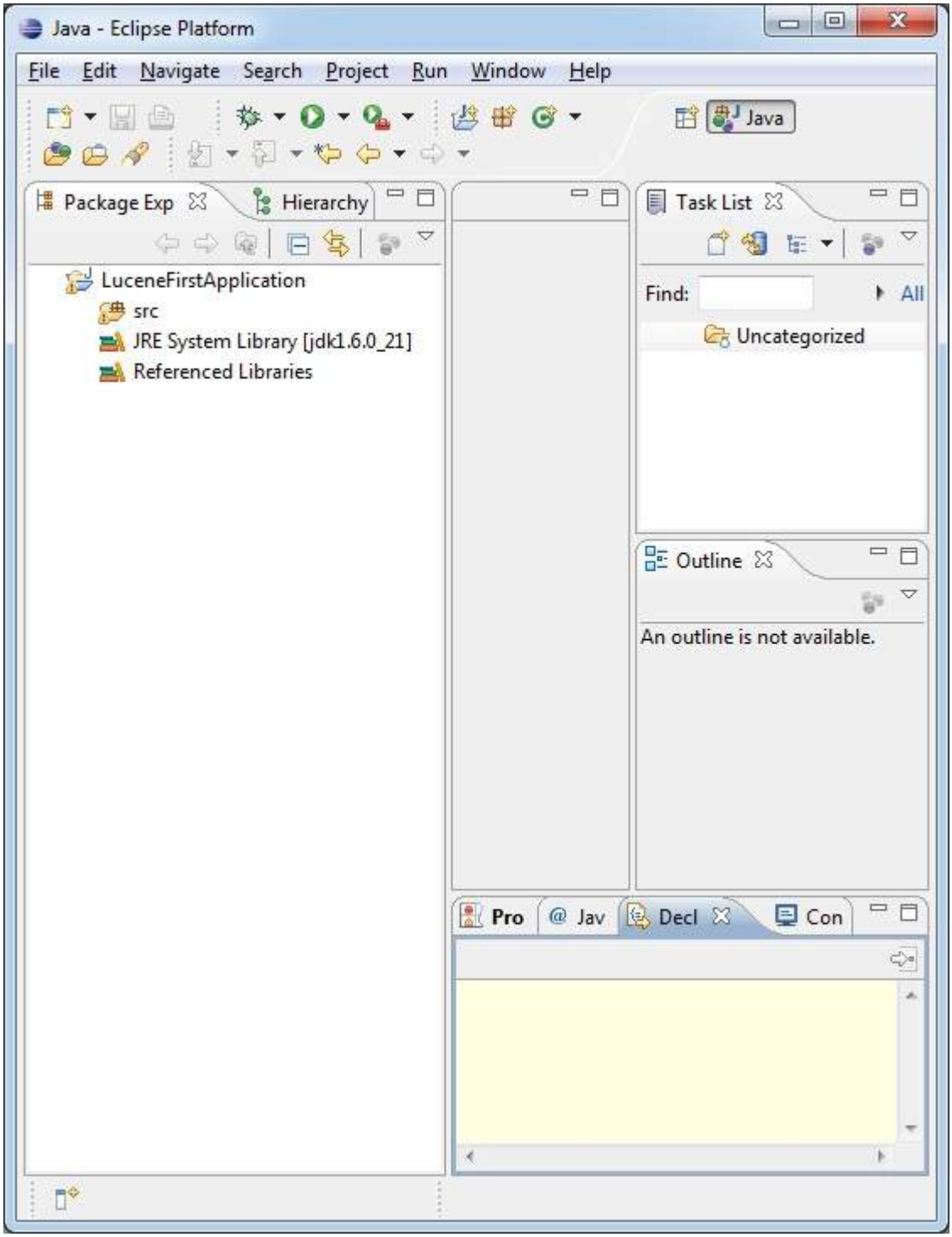
So let us proceed to write a simple Search Application which will print number of search result found. We'll also see the list of indexes created during this process.

Step 1 - Create Java Project:

The first step is to create a simple Java Project using Eclipse IDE. Follow the option **File -> New -> Project** and finally select **Java Project** wizard from the wizard list. Now name your project as **LuceneFirstApplication** using the wizard window as follows:

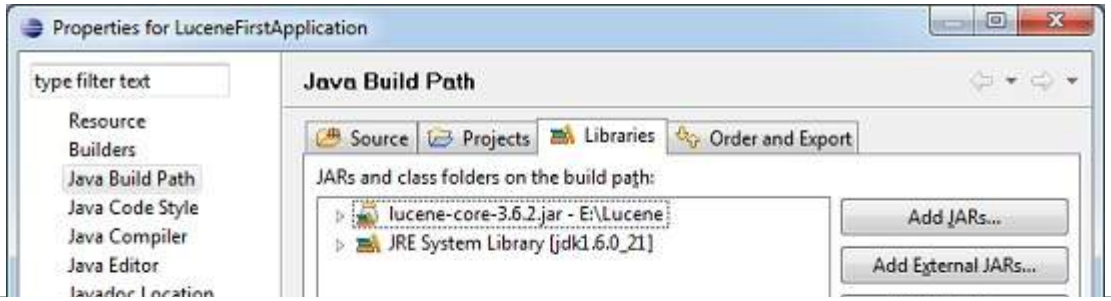


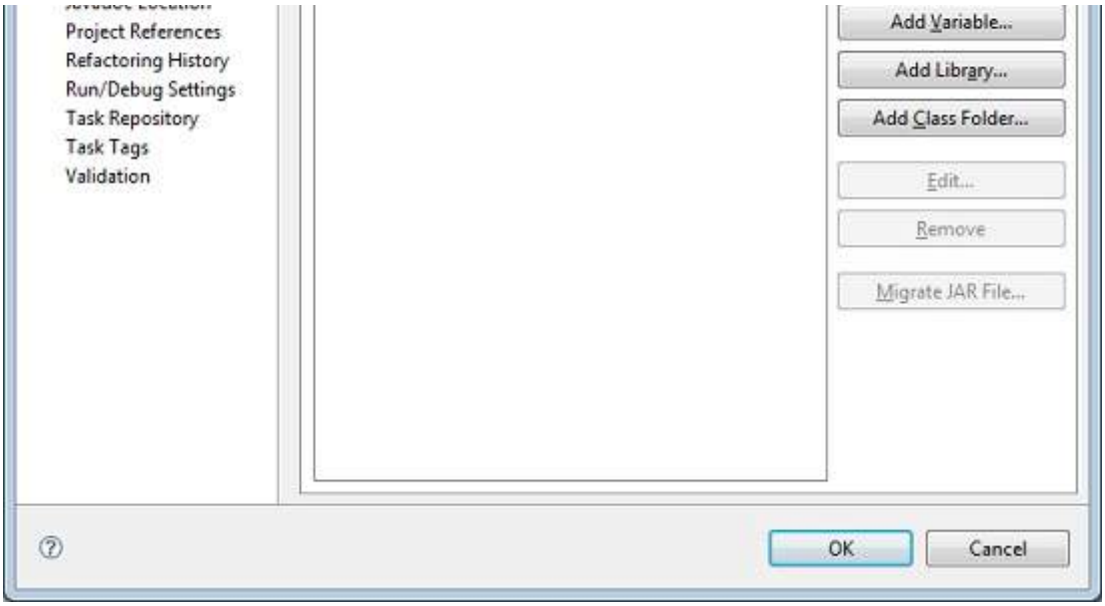
Once your project is created successfully, you will have following content in your **Project Explorer:**



Step 2 - Add Required Libraries:

As a second step let us add Lucene core Framework library in our project. To do this, right click on your project name **LuceneFirstApplication** and then follow the following option available in context menu: **Build Path -> Configure Build Path** to display the Java Build Path window as follows:





Now use **Add External JARs** button available under **Libraries** tab to add the following core JAR from Lucene installation directory:

- lucene-core-3.6.2

Step 3 - Create Source Files:

Now let us create actual source files under the **LuceneFirstApplication** project. First we need to create a package called **com.tutorialspoint.lucene**. To do this, right click on **src** in package explorer section and follow the option : **New -> Package**.

Next we will create **LuceneTester.java** and other java classes under the com.tutorialspoint.lucene package.

LuceneConstants.java

This class is used to provide various constants to be used across the sample application.

```
package com.tutorialspoint.lucene;

public class LuceneConstants {
   public static final String CONTENTS="contents";
   public static final String FILE_NAME="filename";
   public static final String FILE_PATH="filepath";
   public static final int MAX_SEARCH = 10;
}
```

TextFileFilter.java

This class is used as a .txt file filter.

```
package com.tutorialspoint.lucene;

import java.io.File;
import java.io.FileFilter;

public class TextFileFilter implements FileFilter {

   @Override
   public boolean accept(File pathname) {
      return pathname.getName().toLowerCase().endsWith(".txt");
   }
}
```

Indexer.java

This class is used to index the raw data so that we can make it searchable using lucene library.

```
package com.tutorialspoint.lucene;

import java.io.File;
import java.io.FileFilter;
import java.io.FileReader;
import java.io.IOException;

import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.index.IndexWriter;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.FSDirectory;
import org.apache.lucene.util.Version;

public class Indexer {

   private IndexWriter writer;

   public Indexer(String indexDirectoryPath) throws IOException{
      //this directory will contain the indexes
      Directory indexDirectory =
         FSDirectory.open(new File(indexDirectoryPath));

      //create the indexer
      writer = new IndexWriter(indexDirectory,
         new StandardAnalyzer(Version.LUCENE_36),true,
         IndexWriter.MaxFieldLength.UNLIMITED);
   }

   public void close() throws CorruptIndexException, IOException{
      writer.close();
   }

   private Document getDocument(File file) throws IOException{
      Document document = new Document();

      //index file contents
      Field contentField = new Field(LuceneConstants.CONTENTS,
         new FileReader(file));
      //index file name
      Field fileNameField = new Field(LuceneConstants.FILE_NAME,
         file.getName(),
         Field.Store.YES,Field.Index.NOT_ANALYZED);
      //index file path
      Field filePathField = new Field(LuceneConstants.FILE_PATH,
         file.getCanonicalPath(),
         Field.Store.YES,Field.Index.NOT_ANALYZED);

      document.add(contentField);
      document.add(fileNameField);
      document.add(filePathField);

      return document;
   }

   private void indexFile(File file) throws IOException{
      System.out.println("Indexing "+file.getCanonicalPath());
      Document document = getDocument(file);
      writer.addDocument(document);
   }

   public int createIndex(String dataDirPath, FileFilter filter)
      throws IOException{
      //get all files in the data directory
      File[] files = new File(dataDirPath).listFiles();
```

```
      for (File file : files) {
         if(!file.isDirectory()
            && !file.isHidden()
            && file.exists()
            && file.canRead()
            && filter.accept(file)
         ){
            indexFile(file);
         }
      }
      return writer.numDocs();
   }
}
```

Searcher.java

This class is used to search the indexes created by Indexer to search the requested contents.

```
package com.tutorialspoint.lucene;

import java.io.File;
import java.io.IOException;

import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.TopDocs;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.FSDirectory;
import org.apache.lucene.util.Version;

public class Searcher {

   IndexSearcher indexSearcher;
   QueryParser queryParser;
   Query query;

   public Searcher(String indexDirectoryPath)
      throws IOException{
      Directory indexDirectory =
         FSDirectory.open(new File(indexDirectoryPath));
      indexSearcher = new IndexSearcher(indexDirectory);
      queryParser = new QueryParser(Version.LUCENE_36,
         LuceneConstants.CONTENTS,
         new StandardAnalyzer(Version.LUCENE_36));
   }

   public TopDocs search( String searchQuery)
      throws IOException, ParseException{
      query = queryParser.parse(searchQuery);
      return indexSearcher.search(query, LuceneConstants.MAX_SEARCH);
   }

   public Document getDocument(ScoreDoc scoreDoc)
      throws CorruptIndexException, IOException{
      return indexSearcher.doc(scoreDoc.doc);
   }

   public void close() throws IOException{
      indexSearcher.close();
   }
}
```

LuceneTester.java

This class is used to test the indexing and search capability of lucene library.

```
package com.tutorialspoint.lucene;

import java.io.IOException;

import org.apache.lucene.document.Document;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.TopDocs;

public class LuceneTester {

   String indexDir = "E:\\Lucene\\Index";
   String dataDir = "E:\\Lucene\\Data";
   Indexer indexer;
   Searcher searcher;

   public static void main(String[] args) {
      LuceneTester tester;
      try {
         tester = new LuceneTester();
         tester.createIndex();
         tester.search("Mohan");
      } catch (IOException e) {
         e.printStackTrace();
      } catch (ParseException e) {
         e.printStackTrace();
      }
   }

   private void createIndex() throws IOException{
      indexer = new Indexer(indexDir);
      int numIndexed;
      long startTime = System.currentTimeMillis();
      numIndexed = indexer.createIndex(dataDir, new TextFileFilter());
      long endTime = System.currentTimeMillis();
      indexer.close();
      System.out.println(numIndexed+" File indexed, time taken: "
         +(endTime-startTime)+" ms");
   }

   private void search(String searchQuery) throws IOException, ParseException{
      searcher = new Searcher(indexDir);
      long startTime = System.currentTimeMillis();
      TopDocs hits = searcher.search(searchQuery);
      long endTime = System.currentTimeMillis();

      System.out.println(hits.totalHits +
         " documents found. Time :" + (endTime - startTime));
      for(ScoreDoc scoreDoc : hits.scoreDocs) {
         Document doc = searcher.getDocument(scoreDoc);
            System.out.println("File: "
            + doc.get(LuceneConstants.FILE_PATH));
      }
      searcher.close();
   }
}
```

Step 4 - Data & Index directory creation

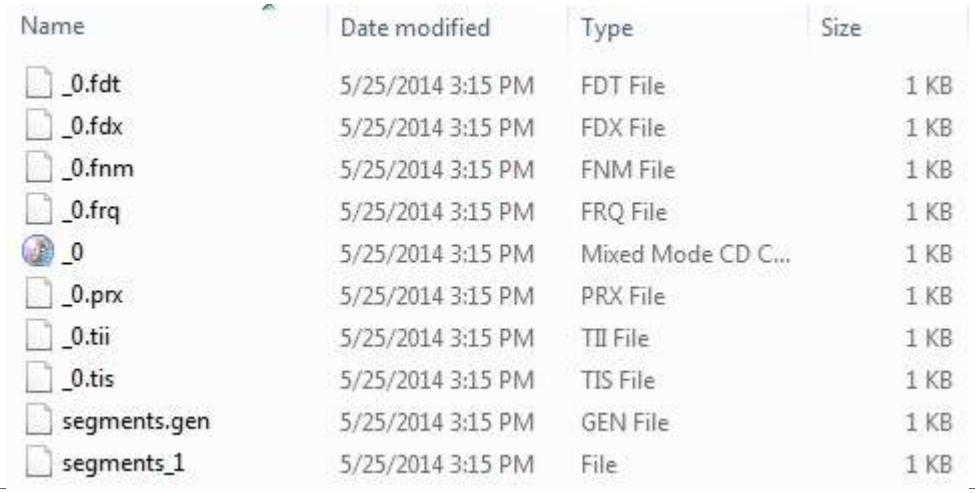
I've used 10 text files named from record1.txt to record10.txt containing simply names and other details of the students and put them in the directory **E:\Lucene\Data.** Test Data. An index directory path should be created as **E:\Lucene\Index**. After running this program, you can see the list of index files created in that folder.

Step 5 - Running the Program:

Once you are done with creating source, creating the raw data, data directory and index directory, you are ready for this step which is compiling and running your program. To do this, Keep LuceneTester.Java file tab active and use either **Run** option available in the Eclipse IDE or use **Ctrl + F11** to compile and run your **LuceneTester** application. If everything is fine with your application, this will print the following message in Eclipse IDE's console:

```
Indexing E:\Lucene\Data\record1.txt
Indexing E:\Lucene\Data\record10.txt
Indexing E:\Lucene\Data\record2.txt
Indexing E:\Lucene\Data\record3.txt
Indexing E:\Lucene\Data\record4.txt
Indexing E:\Lucene\Data\record5.txt
Indexing E:\Lucene\Data\record6.txt
Indexing E:\Lucene\Data\record7.txt
Indexing E:\Lucene\Data\record8.txt
Indexing E:\Lucene\Data\record9.txt
10 File indexed, time taken: 109 ms
1 documents found. Time :0
File: E:\Lucene\Data\record4.txt
```

Once you've run the program successfully, you will have following content in your **index directory:**



Name	Date modified	Type	Size
_0.fdt	5/25/2014 3:15 PM	FDT File	1 KB
_0.fdx	5/25/2014 3:15 PM	FDX File	1 KB
_0.fnm	5/25/2014 3:15 PM	FNM File	1 KB
_0.frq	5/25/2014 3:15 PM	FRQ File	1 KB
_0	5/25/2014 3:15 PM	Mixed Mode CD C...	1 KB
_0.prx	5/25/2014 3:15 PM	PRX File	1 KB
_0.tii	5/25/2014 3:15 PM	TII File	1 KB
_0.tis	5/25/2014 3:15 PM	TIS File	1 KB
segments.gen	5/25/2014 3:15 PM	GEN File	1 KB
segments_1	5/25/2014 3:15 PM	File	1 KB