```
1  public class CSVStoreParameters extends AbstractDataParameters
2    implements DataStoreParameters, FilesystemStoreParameters {
3
4    private static DynClass DYNCLASS;
5
6    private static final String DYNCLASS_NAME = "CSVStoreParameters";
7
8    private static final String FIELD_FILEPATH = "filepath";
9    private static final String FIELD_DELIMITER = "delimiter";
10   private static final String FIELD_CHARSET = "charset";
11   private static final String FIELD_HEADER = "header";
12   private static final String FIELD_HAS_HEADER = "hasHeader";
13   private static final String FIELD_TYPES = "types";
14   private static final String FIELD_ESCAPE_MODE = "escapeMode";
15
16   public static void registerDynClass() {
17     DynObjectManager dynmanager = ToolsLocator.getDynObjectManager();
18     DynClass dynClass;
19     DynField field;
20     if (DYNCLASS == null) {
21       dynClass = dynmanager.add(DYNCLASS_NAME);
22
23       field = dynClass.addDynField(FIELD_FILEPATH);
24       field.setType(DataTypes.STRING);
25       field.setDescription("The path to the file to use as the data source");
26       field.setTheTypeOfAvailableValues(DynField.SINGLE);
27       field.setMandatory(true);
28
29       field = dynClass.addDynField(FIELD_DELIMITER);
30       field.setType(DataTypes.STRING);
31       field.setDescription("The character to use as the column delimiter");
32       field.setDefaultValue(";");
33       field.setTheTypeOfAvailableValues(DynField.SINGLE);
34       field.setMandatory(true);
35
36       field = dynClass.addDynField(FIELD_CHARSET);
37       field.setType(DataTypes.STRING);
38       field.setDescription(" The Charset (encoding) to use while parsing the data.");
39       field.setDefaultValue("ISO-8859-1");
40       field.setTheTypeOfAvailableValues(DynField.SINGLE);
41       field.setMandatory(true);
42
43       field = dynClass.addDynField(FIELD_HEADER);
44       field.setType(DataTypes.STRING);
45       field.setDescription("the first record of data as column headers");
46       field.setTheTypeOfAvailableValues(DynField.SINGLE);
47       field.setMandatory(false);
48
49       field = dynClass.addDynField(FIELD_HAS_HEADER);
50       field.setType(DataTypes.BOOLEAN);
51       field.setDescription("Inform that the first record of data is column headers");
52       field.setTheTypeOfAvailableValues(DynField.SINGLE);
53       field.setMandatory(false);
54
55       field = dynClass.addDynField(FIELD_TYPES);
56       field.setType(DataTypes.STRING);
57       field.setDescription("the types of columns");
58       field.setTheTypeOfAvailableValues(DynField.SINGLE);
```

```
59        field.setMandatory(false);
60
61        field = dynClass.addDynField(FIELD_ESCAPE_MODE);
62        field.setType(DataTypes.INT);
63        field.setDescription("The way to escape an occurance of the text qualifier insid
64        field.setTheTypeOfAvailableValues(DynField.CHOICE);
65        field.setAvailableValues(
66          new DynObjectValueItem[]  {
67            new DynObjectValueItem(
68              new Integer(CsvReader.ESCAPE_MODE_BACKSLASH),
69              "Use a backslash character before the text qualifier to represent an occur
70            ),
71            new DynObjectValueItem(
72              new Integer(CsvReader.ESCAPE_MODE_DOUBLED),
73              "Double up the text qualifier to represent an occurance of the text qualif
74            )
75          }
76        );
77        field.setMandatory(true);
78
79        DYNCLASS = dynClass;
80      }
81    }
82    public String getDataStoreName() {
83      return CSVReadOnlyStoreProvider.NAME;
84    }
85
86    public String getDescription() {
87      return CSVReadOnlyStoreProvider.DESCRIPTION;
88    }
89
90    public boolean isValid() {
91      if (getFileName() == null) {
92              return false;
93      }
94      if (getDelimiter() == null) {
95              return false;
96      }
97      if (getCharsetName() == null) {
98              return false;
99      }
100     return true;
101   }
102
103   public String getFileName() {
104     return (String) getDynValue(FIELD_FILEPATH);
105   }
106
107   public void setFileName(String file) {
108     setDynValue(FIELD_FILEPATH, file);
109   }
110
111   public File getFile() {
112     return new File(this.getFileName());
113   }
114
115   public void setFile(File file) {
116     this.setFileName(file.getPath());
```

```
117   }
118
119   public String getDelimiter() {
120     return (String) getDynValue(FIELD_DELIMITER);
121   }
122
123   public void setDelimiter(String delimiter) {
124     setDynValue(FIELD_DELIMITER, delimiter);
125   }
126
127   public String getCharsetName() {
128     return (String) getDynValue(FIELD_CHARSET);
129   }
130
131   public void setCharsetName(String charsetName) {
132     setDynValue(FIELD_CHARSET, charsetName);
133   }
134
135   public Charset getCharset() {
136     return Charset.forName((String) getDynValue(FIELD_CHARSET));
137   }
138
139   public boolean getHasHeader() {
140     return ((Boolean) getDynValue(FIELD_HAS_HEADER)).booleanValue();
141   }
142
143   public void setHasHeader(boolean hasHeader) {
144     setDynValue(FIELD_HAS_HEADER, new Boolean(hasHeader));
145   }
146
147   public String getHeader() {
148     return (String) getDynValue(FIELD_HEADER);
149   }
150
151   public void setHeader(String header) {
152     setDynValue(FIELD_HEADER, header);
153   }
154
155   public String getTypes() {
156     return (String) getDynValue(FIELD_TYPES);
157   }
158
159   public void setTypes(String types) {
160     setDynValue(FIELD_TYPES, types);
161   }
162
163   public int getEscapeMode() {
164     return ((Integer) getDynValue(FIELD_TYPES)).intValue();
165   }
166
167   public void setEscapeMode(int escapeMode) {
168     setDynValue(FIELD_TYPES, new Integer(escapeMode));
169   }
170
171 }
172
173
174
```

```
175
176
177 public class CSVReadOnlyStoreProvider  extends AbstractMemoryStoreProvider
178   implements ResourceConsumer {
179
180   class CSVData extends ArrayList {
181     /**
182       *
183       */
184     private static final long serialVersionUID = -6386390286245491331L;
185
186     public ArrayList featureTypes = new ArrayList();
187     public FeatureType defaultFeatureType = null;
188   }
189
190   public static final String NAME = "CSV";
191   public static final String DESCRIPTION = "CVS file";
192
193   // Soporte para la DynClass de los metadatos
194   private static DynClass DYNCLASS;
195   private static final String DYNCLASS_NAME = "CSVStore";
196
197
198   protected ResourceProvider resource;
199   protected long OIDCounter;
200
201   protected CSVReadOnlyStoreProvider(DataStoreParameters params,
202     DataStoreProviderServices storeServices)
203     throws InitializeException {
204     super(
205       params,
206       storeServices,
207       ToolsLocator.getDynObjectManager().createDynObject(DYNCLASS)
208     );
209
210     OIDCounter = 0;
211
212     File file = getCSVParameters().getFile();
213     resource = this.createResource(
214       FileResource.NAME,
215       new Object[] { file.getAbsolutePath() }
216     );
217     resource.addConsumer(this);
218
219   }
220
221   public static void registerDynClass() {
222     DynObjectManager dynmanager = ToolsLocator.getDynObjectManager();
223     DynClass dynClass;
224     if (DYNCLASS == null) {
225       dynClass = dynmanager.add(DYNCLASS_NAME);
226
227       DYNCLASS = dynClass;
228     }
229   }
230
231   protected CSVStoreParameters getCSVParameters() {
232     return (CSVStoreParameters) this.getParameters();
```

```
233   }
234
235   public Object createNewOID() {
236     return new Long(++OIDCounter);
237   }
238
239   public String getName() {
240     return NAME;
241   }
242
243   public int getOIDType() {
244     return DataTypes.LONG;
245   }
246
247   public Object getSourceId() {
248     return getCSVParameters().getFileName();
249   }
250
251   public void open() throws OpenException {
252     if (this.data != null) {
253       // El almacen ya ha sido inicializado y sus datos ya estan en memoria,
254       // asi que no hacemos nada.
255       return;
256     }
257
258     CSVData data = (CSVData) resource.getData();
259     if( data != null ) {
260       /*
261         * Se esta volviendo a cargar un fichero que ya esta cargado en
262         * memoria.
263         *
264         * Recuperamos las features y el featuretype del recurso.
265         */
266       getStoreServices().setFeatureTypes(data.featureTypes, data.defaultFeatureType);
267       this.data = data;
268       return;
269     }
270     String resourceName = null;
271     try {
272       resourceName = resource.getName();
273     } catch (AccessResourceException e1) {
274       throw new OpenException(this.getName(), e);
275     }
276     try {
277       resource.begin();
278     } catch (ResourceBeginException e2) {
279       throw new OpenException(resourceName, e2);
280     }
281
282     try {
283       data = new CSVData();
284       this.data = data;
285
286       resource.notifyOpen();
287
288       CsvReader reader = new CsvReader(
289               getCSVParameters().getFileName(),
290               getCSVParameters().getDelimiter().charAt(0),
```

```
291                 getCSVParameters().getCharset()
292       );
293       reader.setEscapeMode(getCSVParameters().getEscapeMode());
294
295       String header = null;
296       if( getCSVParameters().getHasHeader() ) {
297         reader.readHeaders();
298         header = reader.getRawRecord();
299       } else {
300         header = getCSVParameters().getHeader();
301         reader.setHeaders(header.split(getCSVParameters().getDelimiter()));
302       }
303       makeFeatureTypes(
304         data,
305         header,
306         getCSVParameters().getTypes(),
307         getCSVParameters().getDelimiter()
308       );
309
310       getStoreServices().setFeatureTypes(
311         data.featureTypes,
312         data.defaultFeatureType
313       );
314
315       FeatureType featureType;
316       featureType = getStoreServices().getDefaultFeatureType();
317
318       while( reader.readRecord() ) {
319         FeatureProvider feature = this.createFeatureProvider(featureType);
320         Iterator it = featureType.iterator();
321         while( it.hasNext() ) {
322           String name = ((FeatureAttributeDescriptor) it.next()).getName();
323           feature.set(name, reader.get(name));
324         }
325         this.addFeatureProvider(feature);
326       }
327       reader.close();
328       resource.setData(data);
329       resource.notifyClose();
330
331     } catch (FileNotFoundException e) {
332       this.data = null;
333       throw new OpenException(resourceName, e);
334     } catch (IOException e) {
335       this.data = null;
336       throw new OpenException(resourceName, e);
337     } catch (DataException e) {
338       this.data = null;
339       throw new OpenException(resourceName, e);
340     } finally{
341       resource.end();
342     }
343
344   }
345
346   private void makeFeatureTypes(CSVData data, String header, String types, String del
347     EditableFeatureType featureType = null;
348     Map typename2type = new HashMap();
```

```
349     String[] lnames = header.split(delimiter);
350     String[] ltypes = null;
351
352     typename2type.put("string", new Integer(DataTypes.STRING));
353     typename2type.put("boolean", new Integer(DataTypes.BOOLEAN));
354     typename2type.put("byte", new Integer(DataTypes.BYTE));
355     typename2type.put("double", new Integer(DataTypes.DOUBLE));
356     typename2type.put("float", new Integer(DataTypes.FLOAT));
357     typename2type.put("int", new Integer(DataTypes.INT));
358     typename2type.put("long", new Integer(DataTypes.LONG));
359
360     if( types == null ) {
361       ltypes = new String[lnames.length];
362     } else {
363       ltypes = types.split(delimiter);
364       if( ltypes.length != lnames.length ) {
365               throw new IllegalArgumentException();
366       }
367     }
368
369     featureType = getStoreServices().createFeatureType();
370     featureType.setHasOID(true);
371     for( int i=0; i<lnames.length; i++ ) {
372             String typename = ltypes[1];
373             if( typename.contains(":")) {
374                     String[] x = typename.split(":");
375                     int size = Integer.parseInt(x[1]);
376                     int type = ((Integer)typename2type.get(x[0])).intValue();
377                     featureType.add(lnames[i], type, size);
378             } else {
379                     int type = ((Integer)typename2type.get(typename)).intValue();
380                     featureType.add(lnames[i], type);
381             }
382     }
383     data.featureTypes.add(featureType.getNotEditableCopy());
384     data.defaultFeatureType = (FeatureType) (data.featureTypes.get(0));
385   }
386
387 }
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
```

```
407 public class CSVReadWriteStoreProvider extends CSVReadOnlyStoreProvider {
408
409   protected CSVReadWriteStoreProvider(DataStoreParameters params,
410     DataStoreProviderServices storeServices)
411     throws InitializeException {
412     super(
413       params,
414       storeServices
415     );
416   }
417
418   public boolean allowWrite() {
419     return true;
420   }
421
422   public void performChanges(Iterator deleteds, Iterator inserteds,
423     Iterator updateds, Iterator featureTypesChanged) throws PerformEditingException {
424
425     try {
426         resource.begin();
427     } catch (ResourceBeginException e1) {
428         throw new PerformEditingException(this.getName(), e1);
429     }
430
431     String fileName = "";
432     try {
433       CSVData data = (CSVData) this.data;
434       File file = (File) resource.get();
435       fileName = file.getAbsolutePath();
436
437       FeatureSet features = getStoreServices().getFeatureStore().getFeatureSet();
438
439       CsvWriter writer = new CsvWriter(
440         fileName,
441         getCSVParameters().getDelimiter().charAt(0),
442         getCSVParameters().getCharset()
443       );
444
445       writer.setEscapeMode(getCSVParameters().getEscapeMode());
446       writer.setForceQualifier(true);
447
448       String[] names = new String[data.defaultFeatureType.size()];
449       String[] values = new String[names.length];
450
451       Iterator it = data.defaultFeatureType.iterator();
452       for( int i=0; it.hasNext(); i++ ) {
453         names[i] = ((FeatureAttributeDescriptor) it.next()).getName();
454       }
455       writer.writeRecord(names);
456
457       DisposableIterator iterfeatures = features.fastIterator();
458       while (iterfeatures.hasNext()) {
459         Feature feature = (Feature) iterfeatures.next();
460         for( int i=0; i<names.length; i++) {
461                 values[i] = feature.getString(names[i]);
462         }
463         writer.writeRecord(values);
464       }
```

```
465       iterfeatures.dispose();
466
467       features.dispose();
468       resource.notifyChanges();
469
470     } catch (Exception e) {
471       throw new PerformEditingException(fileName, e);
472     } finally {
473       resource.end();
474     }
475   }
476
477 }
478
479
480 public class CSVLibrary extends BaseLibrary {
481
482   protected void doInitialize() throws ReferenceNotRegisteredException {
483
484     CSVStoreParameters.registerDynClass();
485     CSVReadOnlyStoreProvider.registerDynClass();
486
487     DataManagerProviderServices manager = (DataManagerProviderServices) DALLocator.get
488     manager.registerStoreProvider(
489       CSVReadOnlyStoreProvider.NAME,
490       CSVReadWriteStoreProvider.class,
491       CSVStoreParameters.class
492     );
493
494     DALFileLocator.getFilesystemServerExplorerManager().registerProvider(
495       CSVReadOnlyStoreProvider.NAME,
496       CSVReadOnlyStoreProvider.DESCRIPTION,
497       CSVFilesystemServerProvider.class
498     );
499   }
500 }
501
502
503 public class CSVNewStoreParameters extends CSVStoreParameters
504   implements NewFeatureStoreParameters {
505
506   private FeatureType defaultFeatureType = null;
507
508   public FeatureType getDefaultFeatureType() {
509     return this.defaultFeatureType;
510   }
511
512   public void setDefaultFeatureType(FeatureType defaultFeatureType) {
513     this.defaultFeatureType = defaultFeatureType;
514   }
515 }
516
517 public class CSVFilesystemServerProvider implements
518   FilesystemServerExplorerProvider, ResourceConsumer {
519
520   private FilesystemServerExplorerProviderServices serverExplorer = null;
521
522   public boolean canCreate() {
```

```
523     return true;
524   }
525
526   public boolean canCreate(NewDataStoreParameters parameters) {
527     CSVNewStoreParameters params = (CSVNewStoreParameters) parameters;
528     if (params.getFile().getParentFile().canWrite()) {
529       return false;
530     }
531     if (params.getFile().exists()) {
532       if (!params.getFile().canWrite()) {
533             return false;
534       }
535     }
536     if (params.getDefaultFeatureType() == null) {
537       return false;
538     }
539     return true;
540   }
541
542   public void create(NewDataStoreParameters parameters, boolean overwrite)
543     throws CreateException {
544     CSVNewStoreParameters params = (CSVNewStoreParameters) parameters;
545     File file = params.getFile();
546
547     if (file.exists()) {
548       if (overwrite) {
549         if (!file.delete()) {
550           throw new CreateException(
551             this.getDataStoreProviderName(),
552             new IOException("cannot delete file")
553           );
554         }
555       } else {
556         throw new CreateException(
557           this.getDataStoreProviderName(),
558           new IOException("file already exist")
559         );
560       }
561     }
562
563     FileResource resource;
564     try {
565       resource = (FileResource) this.serverExplorer
566         .getServerExplorerProviderServices().createResource(
567           FileResource.NAME,
568           new Object[] { file.getAbsolutePath() }
569       );
570     } catch (InitializeException e1) {
571       throw new CreateException(params.getFileName(), e1);
572     }
573     resource.addConsumer(this);
574
575     try {
576       resource.begin();
577
578       /*
579         * TODO:
580         *
```

```
581         * Escribir las cabeceras en el fichero csv y un comentario con los
582         * tipos de datos de estas.
583         */
584
585       resource.notifyChanges();
586     } catch (Exception e) {
587       throw new CreateException(this.getDataStoreProviderName(), e);
588     } finally {
589       resource.end();
590       resource.removeConsumer(this);
591     }
592   }
593
594   public NewDataStoreParameters getCreateParameters() throws DataException {
595     return new CSVNewStoreParameters();
596   }
597
598   public void initialize(FilesystemServerExplorerProviderServices serverExplorer) {
599     this.serverExplorer = serverExplorer;
600   }
601
602   public void remove(DataStoreParameters parameters) throws RemoveException {
603     CSVStoreParameters params = (CSVStoreParameters) parameters;
604     File file = params.getFile();
605     if (!file.exists()) {
606             throw new RemoveException(this.getDataStoreProviderName(),
607                             new FileNotFoundException(params.getFile()));
608     }
609     if (!file.delete()) {
610             throw new RemoveException(this.getDataStoreProviderName(),
611                             new IOException()); // FIXME Exception
612     }
613   }
614
615   public String getDataStoreProviderName() {
616     return CSVReadOnlyStoreProvider.NAME;
617   }
618
619   public String getDescription() {
620     return CSVReadOnlyStoreProvider.DESCRIPTION;
621   }
622
623   public boolean accept(File pathname) {
624     return (pathname.getName().toLowerCase().endsWith(".csv"));
625   }
626
627   public boolean closeResourceRequested(ResourceProvider resource) {
628     // Do nothing
629     return false;
630   }
631
632   public void resourceChanged(ResourceProvider resource) {
633     // Do nothing
634   }
635
636 }
```