

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      ActivityData.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  * GPL Copyright, The BIRO Project
27:  *
28:  */
29:
30: /*
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36: package export;
37:
38: //-----/
39: //- Imported classes and packages -/
40: //-----/
41: import export.types.ActivityEndReason;
42: import export.types.ActivityStartReason;
43: import org.exolab.castor.xml.Marshaller;
44: import org.exolab.castor.xml.Unmarshaller;
45:
```

```
46: /**
47:  * Class ActivityData.
48:  *
49:  * @version $Revision$ $Date$
50:  */
51: public class ActivityData implements java.io.Serializable {
52:
53:
54:     //-----/
55:     //- Class/Member Variables -/
56:     //-----/
57:     /**
58:     Added by Valentina
59:     */
60:     private int _ID = -1;
61:     /**
62:     * Field _AD_START_DATE.
63:     */
64:     private org.exolab.castor.types.Date _AD_START_DATE;
65:     /**
66:     * Field _hibernateStartDate by Valentina
67:     */
68:     private java.util.Date _hibernateStartDate;
69:     /**
70:     * Field _AD_START_REASON.
71:     */
72:     private ActivityStartReason _AD_START_REASON;
73:     /**
74:     * Field _AD_END_DATE.
75:     */
76:     private org.exolab.castor.types.Date _AD_END_DATE;
77:     /**
78:     * Field _hibernateEndDate by Valentina
79:     */
80:     private java.util.Date _hibernateEndDate;
81:     /**
82:     * Field _AD_END_REASON.
83:     */
84:     private ActivityEndReason _AD_END_REASON;
85:
86:
87:     //-----/
88:     //- Constructors -/
89:     //-----/
90:     public ActivityData() {
```

```
91:         super();
92:     }
93:
94:
95:     //-----/
96:     //- Methods -/
97:     //-----/
98:     //aggiunto da Valentina
99:     /**
100:      * @param vID
101:      */
102:     public void setID(final int vID) {
103:         this._ID = vID;
104:     }
105:
106:     /**
107:      * @param vID
108:      */
109:     public int getID() {
110:         return this._ID;
111:     }
112:
113:     /**
114:      * Returns the value of field 'AD_END_DATE'.
115:      *
116:      * @return the value of field 'EndDate'.
117:      */
118:     public org.exolab.castor.types.Date getAD_END_DATE() {
119:         return this._AD_END_DATE;
120:     }
121:
122:     /** Added by Valentina
123:      * Returns the value of field '_hibernateEndDate'.
124:      *
125:      * @return the value of field '_hibernateEndDate'.
126:      */
127:     public java.util.Date getHibernateEndDate() {
128:         this._hibernateEndDate = this._AD_END_DATE.toDate();
129:         return this._hibernateEndDate;
130:     }
131:
132:     /**
133:      * Returns the value of field 'AD_END_REASON'.
134:      *
135:      * @return the value of field 'AD_END_REASON'.
```

```
136:    */
137:    public ActivityEndReason getAD_END_REASON() {
138:        return this._AD_END_REASON;
139:    }
140:
141:    /**
142:     * Returns the value of field 'AD_START_DATE'.
143:     *
144:     * @return the value of field 'AD_START_DATE'.
145:     */
146:    public org.exolab.castor.types.Date getAD_START_DATE() {
147:        return this._AD_START_DATE;
148:    }
149:
150:    /** Added by Valentina
151:     * Returns the value of field '_hibernateEndDate'.
152:     *
153:     * @return the value of field '_hibernateEndDate'.
154:     */
155:    public java.util.Date getHibernateStartDate() {
156:        this._hibernateStartDate = this._AD_START_DATE.toDate();
157:        return this._hibernateStartDate;
158:    }
159:
160:    /**
161:     * Returns the value of field 'AD_START_REASON'.
162:     *
163:     * @return the value of field 'AD_START_REASON'.
164:     */
165:    public ActivityStartReason getAD_START_REASON() {
166:        return this._AD_START_REASON;
167:    }
168:
169:    /**
170:     * Method isValid.
171:     *
172:     * @return true if this object is valid according to the schema
173:     */
174:    public boolean isValid() {
175:        try {
176:            validate();
177:        } catch (org.exolab.castor.xml.ValidationException vex) {
178:            return false;
179:        }
180:        return true;

```

```
181:     }
182:
183:     /**
184:     *
185:     *
186:     * @param out
187:     * @throws org.exolab.castor.xml.MarshalException if object is
188:     * null or if any SAXException is thrown during marshaling
189:     * @throws org.exolab.castor.xml.ValidationException if this
190:     * object is an invalid instance according to the schema
191:     */
192:     public void marshal(
193:         final java.io.Writer out)
194:         throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
195:         Marshaller.marshal(this, out);
196:     }
197:
198:     /**
199:     *
200:     *
201:     * @param handler
202:     * @throws java.io.IOException if an IOException occurs during
203:     * marshaling
204:     * @throws org.exolab.castor.xml.ValidationException if this
205:     * object is an invalid instance according to the schema
206:     * @throws org.exolab.castor.xml.MarshalException if object is
207:     * null or if any SAXException is thrown during marshaling
208:     */
209:     public void marshal(
210:         final org.xml.sax.ContentHandler handler)
211:         throws java.io.IOException, org.exolab.castor.xml.MarshalException,
org.exolab.castor.xml.ValidationException {
212:         Marshaller.marshal(this, handler);
213:     }
214:
215:     /**
216:     * Sets the value of field 'AD_END_DATE'.
217:     *
218:     * @param AD_END_DATE the value of field 'AD_END_DATE'.
219:     */
220:     public void setAD_END_DATE(
221:         final org.exolab.castor.types.Date AD_END_DATE) {
222:         this._AD_END_DATE = AD_END_DATE;
223:     }
224:
```

```
225:  /** added by Vale
226:   * Sets the value of field 'hibernateEndDate'.
227:   *
228:   * @param AD_END_DATE the value of field 'hibernateEndDate'.
229:   */
230:  public void setHibernateEndDate(
231:      final java.util.Date hibernateEndDate) {
232:      this._hibernateEndDate = hibernateEndDate;
233:  }
234:
235:  /**
236:   * Sets the value of field 'AD_END_REASON'.
237:   *
238:   * @param AD_END_REASON the value of field 'AD_END_REASON'.
239:   */
240:  public void setAD_END_REASON(
241:      final ActivityEndReason AD_END_REASON) {
242:      this._AD_END_REASON = AD_END_REASON;
243:  }
244:
245:  /**
246:   * Sets the value of field 'AD_START_DATE'.
247:   *
248:   * @param AD_START_DATE the value of field 'AD_START_DATE'.
249:   */
250:  public void setAD_START_DATE(
251:      final org.exolab.castor.types.Date AD_START_DATE) {
252:      this._AD_START_DATE = AD_START_DATE;
253:  }
254:
255:  /** added by Vale
256:   * Sets the value of field 'hibernateStartDate'.
257:   *
258:   * @param AD_END_DATE the value of field 'hibernateStartDate'.
259:   */
260:  public void setHibernateStartDate(
261:      final java.util.Date hibernateStartDate) {
262:      this._hibernateStartDate = hibernateStartDate;
263:  }
264:
265:  /**
266:   * Sets the value of field 'AD_START_REASON'.
267:   *
268:   * @param AD_START_REASON the value of field 'AD_START_REASON'.
269:   */
```

```
270: public void setAD_START_REASON(  
271:     final ActivityStartReason AD_START_REASON) {  
272:     this._AD_START_REASON = AD_START_REASON;  
273: }  
274:  
275: /**  
276:  * Method unmarshal.  
277:  *  
278:  * @param reader  
279:  * @throws org.exolab.castor.xml.MarshalException if object is  
280:  * null or if any SAXException is thrown during marshaling  
281:  * @throws org.exolab.castor.xml.ValidationException if this  
282:  * object is an invalid instance according to the schema  
283:  * @return the unmarshaled C:\Documents and  
284:  * Settings\Valentina\Desktop\schema\classes.ActivityData  
285:  */  
286: public static ActivityData unmarshal(  
287:     final java.io.Reader reader)  
288:     throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {  
289:     return (ActivityData) Unmarshaller.unmarshal(ActivityData.class, reader);  
290: }  
291:  
292: /**  
293:  *  
294:  *  
295:  * @throws org.exolab.castor.xml.ValidationException if this  
296:  * object is an invalid instance according to the schema  
297:  */  
298: public void validate()  
299:     throws org.exolab.castor.xml.ValidationException {  
300:     org.exolab.castor.xml.Validator validator = new org.exolab.castor.xml.Validator();  
301:     validator.validate(this);  
302: }  
303: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      DataHeader.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: /**
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36: package export;
37:
38: //-----/
39: //- Imported classes and packages -/
40: //-----/
41: import org.exolab.castor.xml.Marshaller;
42: import org.exolab.castor.xml.Unmarshaller;
43: import export.types.DataSource;
44: /**
45:  * Class SiteHeader.
```



```
46:  *
47:  * @version $Revision$ $Date$
48:  */
49: public class DataHeader implements java.io.Serializable {
50:
51:
52:     //-----/
53:     //- Class/Member Variables -/
54:     //-----/
55:     /**
56:     Aggiunto da Valentina
57:     */
58:     private int _ID    =-1;
59:
60:     /**
61:     * Field _DateCreated.
62:     */
63:     //private org.exolab.castor.types.Date _DateCreated;
64:     /**
65:     * Field _hibernateDateCreated.
66:     */
67:     //private java.util.Date _hibernateDateCreated;
68:
69:     private java.lang.String _dateCreated;
70:
71:
72:     /**
73:     * Unique centre identification number (Defined as a BIRO
74:     * Clinical Site)
75:     */
76:     private DataSource _DS_ID;
77:
78:
79:
80:     //-----/
81:     //- Constructors -/
82:     //-----/
83:     public DataHeader() {
84:         super();
85:     }
86:
87:
88:     //-----/
89:     //- Methods -/
90:     //-----/
```

```
91:  /**
92:   * @param vID
93:   */
94:  public void setID(
95:      final int vID) {
96:      this._ID = vID;
97:  }
98:
99:  /**
100:   * @param vID
101:   */
102:  public int getID() {
103:      return this._ID;
104:  }
105:
106:
107:  /**
108:   * Returns the value of field 'DS_ID'. The field 'DS_ID' has
109:   * the following description: Unique centre identification
110:   * number (Defined as a BIRO Clinical Site)
111:   *
112:   * @return the value of field 'DS_ID'.
113:   */
114:  public DataSource getDS_ID() {
115:      return this._DS_ID;
116:  }
117:
118:  /**
119:   * Returns the value of field 'dateHeaderInformationChecked'.
120:   *
121:   * @return the value of field 'DateHeaderInformationChecked'.
122:   */
123:  /**
124:   public org.exolab.castor.types.Date getDateCreated() {
125:       return this._DateCreated;
126:   }
127:  */
128:  /**
129:   * Returns the value of field '_hibernateDateCreated'.
130:   *
131:   * @return the value of field '_hibernateDateCreated'.
132:   */
133:  /**
134:   public java.util.Date getHibernateDateCreated() {
135:       this._hibernateDateCreated = this._DateCreated.toDate();
```

```
136:         return this._hibernateDateCreated;
137:     }
138: */
139: public java.lang.String getDateCreated() {
140:     return this._dateCreated;
141: }
142: /**
143:  * Method isValid.
144:  *
145:  * @return true if this object is valid according to the schema
146:  */
147: public boolean isValid() {
148:     try {
149:         validate();
150:     } catch (org.exolab.castor.xml.ValidationException vex) {
151:         System.out.println("DataHeaderNotValid");
152:         return false;
153:     }
154:     return true;
155: }
156:
157: /**
158:  *
159:  *
160:  * @param out
161:  * @throws org.exolab.castor.xml.MarshalException if object is
162:  * null or if any SAXException is thrown during marshaling
163:  * @throws org.exolab.castor.xml.ValidationException if this
164:  * object is an invalid instance according to the schema
165:  */
166: public void marshal(
167:     final java.io.Writer out)
168:     throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
169:     Marshaller.marshal(this, out);
170: }
171:
172: /**
173:  *
174:  *
175:  * @param handler
176:  * @throws java.io.IOException if an IOException occurs during
177:  * marshaling
178:  * @throws org.exolab.castor.xml.ValidationException if this
179:  * object is an invalid instance according to the schema
180:  * @throws org.exolab.castor.xml.MarshalException if object is
```

```
181:      * null or if any SAXException is thrown during marshaling
182:      */
183:      public void marshal(
184:          final org.xml.sax.ContentHandler handler)
185:          throws java.io.IOException, org.exolab.castor.xml.MarshalException,
org.exolab.castor.xml.ValidationException {
186:          Marshaller.marshal(this, handler);
187:      }
188:
189:
190:
191:      /**
192:       * Sets the value of field 'DS_ID'. The field 'DS_ID' has the
193:       * following description: Unique centre identification number
194:       * (Defined as a BIRO Clinical Site)
195:       *
196:       * @param DS_ID the value of field 'DS_ID'.
197:       */
198:      public void setDS_ID(
199:          final DataSource DS_ID) {
200:          this._DS_ID = DS_ID;
201:      }
202:
203:
204:      /**
205:       * Sets the value of field 'hibernateDateHeaderInformationChecked'.
206:       *
207:       * @param hibernateDateHeaderInformationChecked the value of field
208:       * 'hibernateDateHeaderInformationChecked'.
209:       */
210:      /*public void setHibernateDateCreated(
211:          final java.util.Date hibernateDateCreated) {
212:          this._hibernateDateCreated = hibernateDateCreated;
213:      }
214:      */
215:      /**
216:       * Sets the value of field 'dateHeaderInformationChecked'.
217:       *
218:       * @param dateHeaderInformationChecked the value of field
219:       * 'dateHeaderInformationChecked'.
220:       */
221:      /*public void setDateCreated(
222:          final org.exolab.castor.types.Date dateCreated) {
223:          this._DateCreated = dateCreated;
224:      }
```

```
225: */
226: public void setDateCreated(final java.lang.String dateCreated) {
227:     this._dateCreated = dateCreated;
228: }
229:
230: /**
231:  * Method unmarshal.
232:  *
233:  * @param reader
234:  * @throws org.exolab.castor.xml.MarshalException if object is
235:  * null or if any SAXException is thrown during marshaling
236:  * @throws org.exolab.castor.xml.ValidationException if this
237:  * object is an invalid instance according to the schema
238:  * @return the unmarshaled DataSourceExport.SiteHeader
239:  */
240: public static export.DataHeader unmarshal(
241:     final java.io.Reader reader)
242:     throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
243:     return (export.DataHeader) Unmarshaller.unmarshal(export.DataHeader.class, reader);
244: }
245:
246: /**
247:  *
248:  *
249:  * @throws org.exolab.castor.xml.ValidationException if this
250:  * object is an invalid instance according to the schema
251:  */
252: public void validate()
253:     throws org.exolab.castor.xml.ValidationException {
254:     org.exolab.castor.xml.Validator validator = new org.exolab.castor.xml.Validator();
255:     validator.validate(this);
256: }
257: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      Data.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: /**
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36:
37: package export;
38:
39:  //-----/
40:  //- Imported classes and packages -/
41:  //-----/
42:
43: import org.exolab.castor.xml.Marshaller;
44: import org.exolab.castor.xml.Unmarshaller;
45: import export.types.*;
```

```
46:
47: /**
48:  * Data corresponding to the patient episode
49:  *
50:  * @version $Revision$ $Date$
51:  */
52: public class Data implements java.io.Serializable {
53:
54:
55:     //-----/
56:     //- Class/Member Variables -/
57:     //-----/
58:     /**
59:     Aggiunto da Valentina
60:     */
61:     private int _ID  =-1;
62:
63:     /**
64:     * Standard BIRO field name
65:     */
66:     private BIRODataSet _episodeFieldName;
67:
68:     /**
69:     * Field result
70:     */
71:     private java.lang.String _episodeFieldValue;
72:
73:
74:     //-----/
75:     //- Constructors -/
76:     //-----/
77:
78:     public Data() {
79:         super();
80:     }
81:
82:
83:     //-----/
84:     //- Methods -/
85:     //-----/
86:
87:     //aggiunto da Valentina
88:     /**
89:     * @param vID
90:     */
```

```
91: public void setID(
92:     final int vID)
93: {
94:     this._ID = vID;
95: }
96:
97: /**
98:  * @param vID
99:  */
100: public int getID()
101: {
102:     return this._ID;
103: }
104:
105: /**
106:  * Returns the value of field 'episodeFieldName'. The field
107:  * 'episodeFieldName' has the following description: Standard
108:  * BIRO field name
109:  *
110:  * @return the value of field 'EpisodeFieldName'.
111:  */
112: public BIRODataSet getEpisodeFieldName(
113: ) {
114:     return this._episodeFieldName;
115: }
116:
117: /**
118:  * Returns the value of field 'episodeFieldValue'. The field
119:  * 'episodeFieldValue' has the following description: Field
120:  * result
121:  *
122:  * @return the value of field 'EpisodeFieldValue'.
123:  */
124: public java.lang.String getEpisodeFieldValue(
125: ) {
126:     return this._episodeFieldValue;
127: }
128:
129: /**
130:  * Method isValid.
131:  *
132:  * @return true if this object is valid according to the schema
133:  */
134: public boolean isValid(
135: ) {
```



```
136:         try {
137:             validate();
138:         } catch (org.exolab.castor.xml.ValidationException vex) {
139:             return false;
140:         }
141:         return true;
142:     }
143:
144:     /**
145:     *
146:     *
147:     * @param out
148:     * @throws org.exolab.castor.xml.MarshalException if object is
149:     * null or if any SAXException is thrown during marshaling
150:     * @throws org.exolab.castor.xml.ValidationException if this
151:     * object is an invalid instance according to the schema
152:     */
153:     public void marshal(
154:         final java.io.Writer out)
155:     throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
156:         Marshaller.marshal(this, out);
157:     }
158:
159:     /**
160:     *
161:     *
162:     * @param handler
163:     * @throws java.io.IOException if an IOException occurs during
164:     * marshaling
165:     * @throws org.exolab.castor.xml.ValidationException if this
166:     * object is an invalid instance according to the schema
167:     * @throws org.exolab.castor.xml.MarshalException if object is
168:     * null or if any SAXException is thrown during marshaling
169:     */
170:     public void marshal(
171:         final org.xml.sax.ContentHandler handler)
172:     throws java.io.IOException, org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
173:         Marshaller.marshal(this, handler);
174:     }
175:
176:     /**
177:     * Sets the value of field 'episodeFieldName'. The field
178:     * 'episodeFieldName' has the following description: Standard
179:     * BIRO field name
180:     *
```

```
181:      * @param episodeFieldName the value of field 'episodeFieldName'
182:      */
183:  public void setEpisodeFieldName(
184:      final BIRODataSet episodeFieldName) {
185:      this._episodeFieldName = episodeFieldName;
186:  }
187:
188:  /**
189:   * Sets the value of field 'episodeFieldValue'. The field
190:   * 'episodeFieldValue' has the following description: Field
191:   * result
192:   *
193:   * @param episodeFieldValue the value of field
194:   * 'episodeFieldValue'.
195:   */
196:  public void setEpisodeFieldValue(
197:      final java.lang.String episodeFieldValue) {
198:      this._episodeFieldValue = episodeFieldValue;
199:  }
200:
201:  /**
202:   * Method unmarshal.
203:   *
204:   * @param reader
205:   * @throws org.exolab.castor.xml.MarshalException if object is
206:   * null or if any SAXException is thrown during marshaling
207:   * @throws org.exolab.castor.xml.ValidationException if this
208:   * object is an invalid instance according to the schema
209:   * @return the unmarshaled dataExport.Data
210:   */
211:  public static Data unmarshal(
212:      final java.io.Reader reader)
213:  throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
214:      return (Data) Unmarshaller.unmarshal(Data.class, reader);
215:  }
216:
217:  /**
218:   *
219:   *
220:   * @throws org.exolab.castor.xml.ValidationException if this
221:   * object is an invalid instance according to the schema
222:   */
223:  public void validate(
224:  )
225:  throws org.exolab.castor.xml.ValidationException {
```

```
226:         org.exolab.castor.xml.Validator validator = new org.exolab.castor.xml.Validator();
227:         validator.validate(this);
228:     }
229:
230: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      ECDataExport.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  **/
30: /*
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36: package export;
37:
38: //-----/
39: //- Imported classes and packages -/
40: //-----/
41: import org.exolab.castor.xml.Marshaller;
42: import org.exolab.castor.xml.Unmarshaller;
43:
44: /**
45:  * Created by Douglas Boyle, 2006-02-16 Each participating client
```

```
46:  * database / system / clinic / data source has a definitionAmended
47:  * by Scott Cunningham, 2007-03-23, version 0.2Split data source
48:  * data from clinical dataAdded extra data fields existing in BIRO
49:  * WP3 datasetAmended by Scott Cunningham, 2007-05-24, version
50:  * 0.3Removal of duplicate element namesAmended by Scott
51:  * Cunningham, 2007-07-24, version 0.4Updated during Data
52:  * Dictionary developments
53:  *
54:  * @version $Revision$ $Date$
55:  */
56: public class ECDataExport implements java.io.Serializable {
57:
58:
59:     //-----/
60:     //- Class/Member Variables -/
61:     //-----/
62:     /**
63:     Aggiunto da Valentina
64:     */
65:     private int _ID = -1;
66:
67:     /**
68:     * Field _patient.
69:     */
70:     private export.Patient _patient;
71:
72:     private export.DataHeader _dataHeader;
73:
74:
75:     //-----/
76:     //- Constructors -/
77:     //-----/
78:     public ECDataExport() {
79:         super();
80:     }
81:
82:
83:     //-----/
84:     //- Methods -/
85:     //-----/
86:     /**
87:     * @param vID
88:     */
89:     public void setID(
90:         final int vID) {
```

```
91:         this._ID = vID;
92:     }
93:
94:     /**
95:      * @param vID
96:      */
97:     public int getID() {
98:         return this._ID;
99:     }
100:
101:     /**
102:      * Method getPatient.
103:      */
104:     public export.Patient getPatient() {
105:         return this._patient;
106:     }
107:
108:     /**
109:      * Method getDataHeader.
110:      */
111:     public export.DataHeader getDataHeader() {
112:         return this._dataHeader;
113:     }
114:
115:
116:
117:     /**
118:      * Method isValid.
119:      *
120:      * @return true if this object is valid according to the schema
121:      */
122:     public boolean isValid() {
123:         try {
124:             validate();
125:         } catch (org.exolab.castor.xml.ValidationException vex) {
126:             return false;
127:         }
128:         return true;
129:     }
130:
131:     /**
132:      *
133:      *
134:      * @param out
135:      * @throws org.exolab.castor.xml.MarshalException if object is
```

```
136:      * null or if any SAXException is thrown during marshaling
137:      * @throws org.exolab.castor.xml.ValidationException if this
138:      * object is an invalid instance according to the schema
139:      */
140:  public void marshal(
141:      final java.io.Writer out)
142:      throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
143:      Marshaller.marshal(this, out);
144:  }
145:
146:  /**
147:   *
148:   *
149:   * @param handler
150:   * @throws java.io.IOException if an IOException occurs during
151:   * marshaling
152:   * @throws org.exolab.castor.xml.ValidationException if this
153:   * object is an invalid instance according to the schema
154:   * @throws org.exolab.castor.xml.MarshalException if object is
155:   * null or if any SAXException is thrown during marshaling
156:   */
157:  public void marshal(
158:      final org.xml.sax.ContentHandler handler)
159:      throws java.io.IOException, org.exolab.castor.xml.MarshalException,
org.exolab.castor.xml.ValidationException {
160:      Marshaller.marshal(this, handler);
161:  }
162:
163:  /**
164:   *
165:   *
166:   * @param vPatient
167:   */
168:  public void setPatient( export.Patient vPatient) {
169:
170:      this._patient=vPatient;
171:  }
172:
173:  /**
174:   *
175:   *
176:   * @param vDataHeader
177:   */
178:  public void setDataHeader( export.DataHeader vDataHeader) {
179:
```

```
180:         this._dataHeader=vDataHeader;
181:     }
182:
183:     /**
184:     * Method unmarshal.
185:     *
186:     * @param reader
187:     * @throws org.exolab.castor.xml.MarshalException if object is
188:     * null or if any SAXException is thrown during marshaling
189:     * @throws org.exolab.castor.xml.ValidationException if this
190:     * object is an invalid instance according to the schema
191:     * @return the unmarshaled export.ECDataExport
192:     */
193:     public static export.ECDataExport unmarshal(
194:         final java.io.Reader reader)
195:         throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
196:         return (export.ECDataExport) Unmarshaller.unmarshal(export.ECDataExport.class, reader);
197:     }
198:
199:     /**
200:     *
201:     *
202:     * @throws org.exolab.castor.xml.ValidationException if this
203:     * object is an invalid instance according to the schema
204:     */
205:     public void validate()
206:         throws org.exolab.castor.xml.ValidationException {
207:         org.exolab.castor.xml.Validator validator = new org.exolab.castor.xml.Validator();
208:         validator.validate(this);
209:     }
210:
211:
212:
213: }
```



```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      ECDataSourceExport.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  **/
30: /**
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36:
37: package export;
38:
39:  //-----/
40:  //- Imported classes and packages -/
41:  //-----/
42:
43: import org.exolab.castor.xml.Marshaller;
44: import org.exolab.castor.xml.Unmarshaller;
45:
```

```
46: /**
47:  * Created by Douglas Boyle, 2006-02-16Each participating client
48:  * database / system / clinic has a definitionAmended by Scott
49:  * Cunningham, 2007-03-23, version 0.2Split data source data from
50:  * clinical dataAdded extra data fields existing in BIRO WP3
51:  * datasetAmended by Scott Cunningham, 2007-05-24, version
52:  * 0.3Removal of duplicate element namesAmended by Scott
53:  * Cunningham, 2007-07-24, version 0.4Updated during Data
54:  * Dictionary development
55:  *
56:  * @version $Revision$ $Date$
57:  */
58: public class ECDataSourceExport implements java.io.Serializable {
59:
60:
61:     //-----/
62:     //- Class/Member Variables -/
63:     //-----/
64:     /**
65:     Aggiunto da Valentina
66:     */
67:     private int _ID =-1;
68:
69:     /**
70:     * Field _siteProfile.
71:     */
72:     private export.SiteProfile _siteProfile;
73:
74:     /**
75:     * Field _siteHeader.
76:     */
77:     private export.SiteHeader _siteHeader;
78:
79:
80:     /**
81:     * Field _fieldExportProfilesList.
82:     */
83:     private java.util.List _fieldExportProfilesList;
84:
85:
86:     //-----/
87:     //- Constructors -/
88:     //-----/
89:
90:     public ECDataSourceExport() {
```

```
91:         super();
92:         this._fieldExportProfilesList = new java.util.ArrayList();
93:     }
94:
95:
96:         //-----/
97:         //- Methods -/
98:         //-----/
99:
100:
101:     /**
102:      * @param vID
103:      */
104:     public void setID(
105:         final int vID)
106:     {
107:         this._ID = vID;
108:     }
109:
110:     /**
111:      * @param vID
112:      */
113:     public int getID()
114:     {
115:         return this._ID;
116:     }
117:
118:
119:     /**
120:      *
121:      *
122:      * @param vFieldExportProfiles
123:      * @throws java.lang.IndexOutOfBoundsException if the index
124:      * given is outside the bounds of the collection
125:      */
126:     public void addFieldExportProfiles(
127:         final export.FieldExportProfiles vFieldExportProfiles)
128:     throws java.lang.IndexOutOfBoundsException {
129:         this._fieldExportProfilesList.add(vFieldExportProfiles);
130:     }
131:
132:     /**
133:      *
134:      *
135:      * @param index
```

```
136:      * @param vFieldExportProfiles
137:      * @throws java.lang.IndexOutOfBoundsException if the index
138:      * given is outside the bounds of the collection
139:      */
140: public void addFieldExportProfiles(
141:     final int index,
142:     final export.FieldExportProfiles vFieldExportProfiles)
143: throws java.lang.IndexOutOfBoundsException {
144:     this._fieldExportProfilesList.add(index, vFieldExportProfiles);
145: }
146:
147: /**
148:  * Method enumerateFieldExportProfiles.
149:  *
150:  * @return an Enumeration over all possible elements of this
151:  * collection
152:  */
153: public java.util Enumeration enumerateFieldExportProfiles(
154: ) {
155:     return java.util.Collections.enumeration(this._fieldExportProfilesList);
156: }
157:
158: /**
159:  * Method getFieldExportProfiles.
160:  *
161:  * @param index
162:  * @throws java.lang.IndexOutOfBoundsException if the index
163:  * given is outside the bounds of the collection
164:  * @return the value of the
165:  * dataSourceExport.FieldExportProfiles at the given index
166:  */
167: public export.FieldExportProfiles getFieldExportProfiles(
168:     final int index)
169: throws java.lang.IndexOutOfBoundsException {
170:     // check bounds for index
171:     if (index < 0 || index >= this._fieldExportProfilesList.size()) {
172:         throw new IndexOutOfBoundsException("getFieldExportProfiles: Index value '" + index + "' not in range
[0.." + (this._fieldExportProfilesList.size() - 1) + "]");
173:     }
174:
175:     return (export.FieldExportProfiles) _fieldExportProfilesList.get(index);
176: }
177:
178: /**
179:  * Method getFieldExportProfiles.Returns the contents of the
```

```
180:      * collection in an Array. <p>Note: Just in case the
181:      * collection contents are changing in another thread, we pass
182:      * a 0-length Array of the correct type into the API call.
183:      * This way we <i>know</i> that the Array returned is of
184:      * exactly the correct length.
185:      *
186:      * @return this collection as an Array
187:      */
188: public export.FieldExportProfiles[] getFieldExportProfiles(
189: ) {
190:     export.FieldExportProfiles[] array = new export.FieldExportProfiles[0];
191:     return (export.FieldExportProfiles[]) this._fieldExportProfilesList.toArray(array);
192: }
193:
194: /**
195:  * Method getFieldExportProfilesCount.
196:  *
197:  * @return the size of this collection
198:  */
199: public int getFieldExportProfilesCount(
200: ) {
201:     return this._fieldExportProfilesList.size();
202: }
203:
204:
205: /**
206:  * Returns the value of field 'siteHeader'.
207:  *
208:  * @return the value of field 'SiteHeader'.
209:  */
210: public export.SiteHeader getSiteHeader(
211: ) {
212:     return this._siteHeader;
213: }
214:
215: /**
216:  * Returns the value of field 'siteProfile'.
217:  *
218:  * @return the value of field 'SiteProfile'.
219:  */
220: public export.SiteProfile getSiteProfile(
221: ) {
222:     return this._siteProfile;
223: }
224:
```

```
225:
226: /**
227:  * Method isValid.
228:  *
229:  * @return true if this object is valid according to the schema
230:  */
231: public boolean isValid(
232: ) {
233:     try {
234:         validate();
235:     } catch (org.exolab.castor.xml.ValidationException vex) {
236:         return false;
237:     }
238:     return true;
239: }
240:
241: /**
242:  * Method iterateFieldExportProfiles.
243:  *
244:  * @return an Iterator over all possible elements in this
245:  * collection
246:  */
247: public java.util.Iterator iterateFieldExportProfiles(
248: ) {
249:     return this._fieldExportProfilesList.iterator();
250: }
251:
252: /**
253:  *
254:  *
255:  * @param out
256:  * @throws org.exolab.castor.xml.MarshalException if object is
257:  * null or if any SAXException is thrown during marshaling
258:  * @throws org.exolab.castor.xml.ValidationException if this
259:  * object is an invalid instance according to the schema
260:  */
261: public void marshal(
262:     final java.io.Writer out)
263: throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
264:     Marshaller.marshal(this, out);
265: }
266:
267: /**
268:  *
269:  *
```

```
270:      * @param handler
271:      * @throws java.io.IOException if an IOException occurs during
272:      * marshaling
273:      * @throws org.exolab.castor.xml.ValidationException if this
274:      * object is an invalid instance according to the schema
275:      * @throws org.exolab.castor.xml.MarshalException if object is
276:      * null or if any SAXException is thrown during marshaling
277:      */
278:      public void marshal(
279:          final org.xml.sax.ContentHandler handler)
280:          throws java.io.IOException, org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
281:          Marshaller.marshal(this, handler);
282:      }
283:
284:      /**
285:      */
286:      public void removeAllFieldExportProfiles(
287:      ) {
288:          this._fieldExportProfilesList.clear();
289:      }
290:
291:      /**
292:      * Method removeFieldExportProfiles.
293:      *
294:      * @param vFieldExportProfiles
295:      * @return true if the object was removed from the collection.
296:      */
297:      public boolean removeFieldExportProfiles(
298:          final export.FieldExportProfiles vFieldExportProfiles) {
299:          boolean removed = _fieldExportProfilesList.remove(vFieldExportProfiles);
300:          return removed;
301:      }
302:
303:      /**
304:      * Method removeFieldExportProfilesAt.
305:      *
306:      * @param index
307:      * @return the element removed from the collection
308:      */
309:      public export.FieldExportProfiles removeFieldExportProfilesAt(
310:          final int index) {
311:          java.lang.Object obj = this._fieldExportProfilesList.remove(index);
312:          return (export.FieldExportProfiles) obj;
313:      }
314:
```

```
315:  /**
316:  *
317:  *
318:  * @param index
319:  * @param vFieldExportProfiles
320:  * @throws java.lang.IndexOutOfBoundsException if the index
321:  * given is outside the bounds of the collection
322:  */
323:  public void setFieldExportProfiles(
324:      final int index,
325:      final export.FieldExportProfiles vFieldExportProfiles)
326:  throws java.lang.IndexOutOfBoundsException {
327:      // check bounds for index
328:      if (index < 0 || index >= this._fieldExportProfilesList.size()) {
329:          throw new IndexOutOfBoundsException("setFieldExportProfiles: Index value '" + index + "' not in range
[0.." + (this._fieldExportProfilesList.size() - 1) + "]");
330:      }
331:
332:      this._fieldExportProfilesList.set(index, vFieldExportProfiles);
333:  }
334:
335:  /**
336:  *
337:  *
338:  * @param vFieldExportProfilesArray
339:  */
340:  public void setFieldExportProfiles(
341:      final export.FieldExportProfiles[] vFieldExportProfilesArray) {
342:      //-- copy array
343:      _fieldExportProfilesList.clear();
344:
345:      for (int i = 0; i < vFieldExportProfilesArray.length; i++) {
346:          this._fieldExportProfilesList.add(vFieldExportProfilesArray[i]);
347:      }
348:  }
349:
350:  /**
351:  * Sets the value of field 'siteHeader'.
352:  *
353:  * @param siteHeader the value of field 'siteHeader'.
354:  */
355:  public void setSiteHeader(
356:      final export.SiteHeader siteHeader) {
357:      this._siteHeader = siteHeader;
358:  }
```



```
359:
360: /**
361:  * Sets the value of field 'siteProfile'.
362:  *
363:  * @param siteProfile the value of field 'siteProfile'.
364:  */
365: public void setSiteProfile(
366:     final export.SiteProfile siteProfile) {
367:     this._siteProfile = siteProfile;
368: }
369:
370:
371: /**
372:  * Method unmarshal.
373:  *
374:  * @param reader
375:  * @throws org.exolab.castor.xml.MarshalException if object is
376:  * null or if any SAXException is thrown during marshaling
377:  * @throws org.exolab.castor.xml.ValidationException if this
378:  * object is an invalid instance according to the schema
379:  * @return the unmarshaled dataSourceExport.ECDataSourceExport
380:  */
381: public static export.ECDataSourceExport unmarshal(
382:     final java.io.Reader reader)
383: throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
384:     return (export.ECDataSourceExport) Unmarshaller.unmarshal(export.ECDataSourceExport.class, reader);
385: }
386:
387: /**
388:  *
389:  *
390:  * @throws org.exolab.castor.xml.ValidationException if this
391:  * object is an invalid instance according to the schema
392:  */
393: public void validate(
394: )
395: throws org.exolab.castor.xml.ValidationException {
396:     org.exolab.castor.xml.Validator validator = new org.exolab.castor.xml.Validator();
397:     validator.validate(this);
398: }
399:
400: /**
401:  * Method getFieldExportProfilesList (richiesto da Hibernate)
402:  */
403: public java.util.List getFieldExportProfilesList(
```

```
404:     ) {
405:         return this._fieldExportProfilesList;
406:     }
407:
408:     /**
409:      * Method setFieldExportProfilesList uguale a setFieldExportProfile (richiesto da Hibernate)
410:      */
411:     public void setFieldExportProfilesList(
412:         java.util.List fieldExportProfilesList) {
413:         this._fieldExportProfilesList= fieldExportProfilesList;
414:     }
415:
416:
417: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      EpisodeData.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: /*
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36:
37: package export;
38:
39:  //-----/
40:  //- Imported classes and packages -/
41:  //-----/
42:
43: import export.Data;
44: import org.exolab.castor.xml.Marshaller;
45: import org.exolab.castor.xml.Unmarshaller;
```

```
46:
47: /**
48:  * Patients have events that happen chrologocally (patient
49:  * episodes)
50:  *
51:  * @version $Revision$ $Date$
52:  */
53: public class EpisodeData implements java.io.Serializable {
54:
55:
56:     //-----/
57:     //- Class/Member Variables -/
58:     //-----/
59:     /**
60:     Aggiunto da Valentina
61:     */
62:     private int _ID =-1;
63:
64:     /**
65:     * Date of the patient episode
66:     */
67:     private org.exolab.castor.types.Date _episodeDate;
68:
69:     /**
70:     * Date of the patient episode by Valentina
71:     */
72:     private java.util.Date _hibernateEpisodeDate;
73:
74:     /**
75:     * Data corresponding to the patient episode
76:     */
77:     private java.util.List _dataList;
78:
79:
80:     //-----/
81:     //- Constructors -/
82:     //-----/
83:
84:     public EpisodeData() {
85:         super();
86:         this._dataList = new java.util.ArrayList();
87:     }
88:
89:
90:     //-----/
```

```
91:      //- Methods -/  
92:      //-----/  
93:  
94:      //aggiunto da Valentina  
95:      /**  
96:       * @param vID  
97:       */  
98:      public void setID(  
99:          final int vID)  
100:      {  
101:          this._ID = vID;  
102:      }  
103:  
104:      /**  
105:       * @param vID  
106:       */  
107:      public int getID()  
108:      {  
109:          return this._ID;  
110:      }  
111:  
112:      /**  
113:       *  
114:       *  
115:       * @param vData  
116:       * @throws java.lang.IndexOutOfBoundsException if the index  
117:       * given is outside the bounds of the collection  
118:       */  
119:      public void addData(  
120:          final Data vData)  
121:      throws java.lang.IndexOutOfBoundsException {  
122:          this._dataList.add(vData);  
123:      }  
124:  
125:      /**  
126:       *  
127:       *  
128:       * @param index  
129:       * @param vData  
130:       * @throws java.lang.IndexOutOfBoundsException if the index  
131:       * given is outside the bounds of the collection  
132:       */  
133:      public void addData(  
134:          final int index,  
135:          final Data vData)
```

```
136:     throws java.lang.IndexOutOfBoundsException {
137:         this._dataList.add(index, vData);
138:     }
139:
140:     /**
141:      * Method enumerateData.
142:      *
143:      * @return an Enumeration over all possible elements of this
144:      * collection
145:      */
146:     public java.util.Enumeration enumerateData(
147:     ) {
148:         return java.util.Collections.enumeration(this._dataList);
149:     }
150:
151:     /**
152:      * Method getData.
153:      *
154:      * @param index
155:      * @throws java.lang.IndexOutOfBoundsException if the index
156:      * given is outside the bounds of the collection
157:      * @return the value of the dataExport.Data at the given index
158:      */
159:     public Data getData(
160:         final int index)
161:     throws java.lang.IndexOutOfBoundsException {
162:         // check bounds for index
163:         if (index < 0 || index >= this._dataList.size()) {
164:             throw new IndexOutOfBoundsException("getData: Index value '" + index + "' not in range [0.." +
(this._dataList.size() - 1) + "]");
165:         }
166:
167:         return (Data) _dataList.get(index);
168:     }
169:
170:     /**
171:      * Method getData.Returns the contents of the collection in an
172:      * Array. <p>Note: Just in case the collection contents are
173:      * changing in another thread, we pass a 0-length Array of the
174:      * correct type into the API call. This way we <i>know</i>
175:      * that the Array returned is of exactly the correct length.
176:      *
177:      * @return this collection as an Array
178:      */
179:     public Data[] getData(
```

```
180:     ) {
181:         Data[] array = new Data[0];
182:         return (Data[]) this._dataList.toArray(array);
183:     }
184:
185:     /**
186:      * Method getDataCount.
187:      *
188:      * @return the size of this collection
189:      */
190:     public int getDataCount(
191:     ) {
192:         return this._dataList.size();
193:     }
194:
195:     /**
196:      * Returns the value of field 'episodeDate'. The field
197:      * 'episodeDate' has the following description: Date of the
198:      * patient episode
199:      *
200:      * @return the value of field 'EpisodeDate'.
201:      */
202:     public org.exolab.castor.types.Date getEpisodeDate(
203:     ) {
204:         return this._episodeDate;
205:     }
206:
207:     /** Add by Vale
208:      * Returns the value of field '_hibernateEpisodeDate'. The field
209:      * 'episodeDate' has the following description: Date of the
210:      * patient episode
211:      *
212:      * @return the value of field '_hibernateEpisodeDate'.
213:      */
214:     public java.util.Date getHibernateEpisodeDate(
215:     ) {
216:         this._hibernateEpisodeDate = this._episodeDate.toDate();
217:         return this._hibernateEpisodeDate;
218:     }
219:
220:     /**
221:      * Method isValid.
222:      *
223:      * @return true if this object is valid according to the schema
224:      */
```

```
225: public boolean isValid(
226: ) {
227:     try {
228:         validate();
229:     } catch (org.exolab.castor.xml.ValidationException vex) {
230:         return false;
231:     }
232:     return true;
233: }
234:
235: /**
236:  * Method iterateData.
237:  *
238:  * @return an Iterator over all possible elements in this
239:  * collection
240:  */
241: public java.util.Iterator iterateData(
242: ) {
243:     return this._dataList.iterator();
244: }
245:
246: /**
247:  *
248:  *
249:  * @param out
250:  * @throws org.exolab.castor.xml.MarshalException if object is
251:  * null or if any SAXException is thrown during marshaling
252:  * @throws org.exolab.castor.xml.ValidationException if this
253:  * object is an invalid instance according to the schema
254:  */
255: public void marshal(
256:     final java.io.Writer out)
257: throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
258:     Marshaller.marshal(this, out);
259: }
260:
261: /**
262:  *
263:  *
264:  * @param handler
265:  * @throws java.io.IOException if an IOException occurs during
266:  * marshaling
267:  * @throws org.exolab.castor.xml.ValidationException if this
268:  * object is an invalid instance according to the schema
269:  * @throws org.exolab.castor.xml.MarshalException if object is
```



```
270:      * null or if any SAXException is thrown during marshaling
271:      */
272:  public void marshal(
273:      final org.xml.sax.ContentHandler handler)
274:  throws java.io.IOException, org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
275:      Marshaller.marshal(this, handler);
276:  }
277:
278:  /**
279:   */
280:  public void removeAllData(
281:  ) {
282:      this._dataList.clear();
283:  }
284:
285:  /**
286:   * Method removeData.
287:   *
288:   * @param vData
289:   * @return true if the object was removed from the collection.
290:   */
291:  public boolean removeData(
292:      final Data vData) {
293:      boolean removed = _dataList.remove(vData);
294:      return removed;
295:  }
296:
297:  /**
298:   * Method removeDataAt.
299:   *
300:   * @param index
301:   * @return the element removed from the collection
302:   */
303:  public Data removeDataAt(
304:      final int index) {
305:      java.lang.Object obj = this._dataList.remove(index);
306:      return (Data) obj;
307:  }
308:
309:  /**
310:   *
311:   *
312:   * @param index
313:   * @param vData
314:   * @throws java.lang.IndexOutOfBoundsException if the index
```

```
315:      * given is outside the bounds of the collection
316:      */
317:      public void setData(
318:          final int index,
319:          final Data vData)
320:          throws java.lang.IndexOutOfBoundsException {
321:          // check bounds for index
322:          if (index < 0 || index >= this._dataList.size()) {
323:              throw new IndexOutOfBoundsException("setData: Index value '" + index + "' not in range [0.." +
(this._dataList.size() - 1) + "]");
324:          }
325:
326:          this._dataList.set(index, vData);
327:      }
328:
329:      /**
330:       *
331:       *
332:       * @param vDataArray
333:       */
334:      public void setData(
335:          final Data[] vDataArray) {
336:          //-- copy array
337:          _dataList.clear();
338:
339:          for (int i = 0; i < vDataArray.length; i++) {
340:              this._dataList.add(vDataArray[i]);
341:          }
342:      }
343:
344:      /**
345:       * Sets the value of field 'episodeDate'. The field
346:       * 'episodeDate' has the following description: Date of the
347:       * patient episode
348:       *
349:       * @param episodeDate the value of field 'episodeDate'.
350:       */
351:      public void setEpisodeDate(
352:          final org.exolab.castor.types.Date episodeDate) {
353:          this._episodeDate = episodeDate;
354:      }
355:
356:      /** add by Vale
357:       * Sets the value of field 'hibernateEpisodeDate'. The field
358:       * 'hibernateEpisodeDate' has the following description: Date of the
```

```
359:      * patient episode
360:      *
361:      * @param episodeDate the value of field 'hibernateEpisodeDate'.
362:      */
363:      public void setHibernateEpisodeDate(
364:          final java.util.Date hibernateEpisodeDate) {
365:          this._hibernateEpisodeDate = hibernateEpisodeDate;
366:      }
367:
368:      /**
369:       * Method unmarshal.
370:       *
371:       * @param reader
372:       * @throws org.exolab.castor.xml.MarshalException if object is
373:       * null or if any SAXException is thrown during marshaling
374:       * @throws org.exolab.castor.xml.ValidationException if this
375:       * object is an invalid instance according to the schema
376:       * @return the unmarshaled dataExport.EpisodeData
377:       */
378:      public static export.EpisodeData unmarshal(
379:          final java.io.Reader reader)
380:          throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
381:          return (export.EpisodeData) Unmarshaller.unmarshal(export.EpisodeData.class, reader);
382:      }
383:
384:      /**
385:       *
386:       *
387:       * @throws org.exolab.castor.xml.ValidationException if this
388:       * object is an invalid instance according to the schema
389:       */
390:      public void validate(
391:      )
392:      throws org.exolab.castor.xml.ValidationException {
393:          org.exolab.castor.xml.Validator validator = new org.exolab.castor.xml.Validator();
394:          validator.validate(this);
395:      }
396:      /** Aggiunto da Valentina
397:       * Method getDataList (richiesto da Hibernate)
398:       */
399:      public java.util.List getDataList(
400:      ) {
401:          return this._dataList;
402:      }
403:
```

```
404:    /** Aggiunto da Valentina
405:     * Method setDataList uguale a setData (richiesto da Hibernate)
406:     */
407:    public void setDataList(
408:        java.util.List dataList) {
409:        this._dataList= dataList;
410:
411:    }
412:
413: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      Export.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  **/
30: /*
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36:
37: package export;
38:
39:  //-----/
40:  //- Imported classes and packages -/
41:  //-----/
42:
43: import org.exolab.castor.xml.Marshaller;
44: import org.exolab.castor.xml.Unmarshaller;
45:
```

```
46: /**
47:  * Class Export.
48:  *
49:  * @version $Revision$ $Date$
50:  */
51: public class Export implements java.io.Serializable {
52:
53:
54:     //-----/
55:     //- Class/Member Variables -/
56:     //-----/
57:     /*Aggiunto da Valentina
58:      */
59:     private int _ID = -1;
60:
61: /**
62:  * Created by Douglas Boyle, 2006-02-16
63:  */
64: private java.util.List _ECDataExportList;
65:
66: /**
67:  * Created by Douglas Boyle, 2006-02-16
68:  */
69: private export.ECDataSourceExport _ECDataSourceExport;
70:
71:
72:     //-----/
73:     //- Constructors -/
74:     //-----/
75:
76: public Export() {
77:     super();
78:     this._ECDataExportList = new java.util.ArrayList();
79: }
80:
81:
82:     //-----/
83:     //- Methods -/
84:     //-----/
85: /**
86:  * @param vID
87:  */
88: public void setID(
89:     final int vID)
90: {
```

```
91:         this._ID = vID;
92:     }
93:
94:     /**
95:      * @param vID
96:      */
97:     public int getID()
98:     {
99:         return this._ID;
100:    }
101:
102:    /**
103:     * Sets the value of field 'ECDataSourceExport'. The field
104:     * 'ECDataSourceExport' has the following description: Created
105:     * by Douglas Boyle, 2006-02-16
106:     *
107:     * @param ECDataSourceExport the value of field
108:     * 'ECDataSourceExport'.
109:     */
110:    public void setECDataSourceExport(
111:        final export.ECDataSourceExport ECDataSourceExport) {
112:        this._ECDataSourceExport = ECDataSourceExport;
113:    }
114:
115:    /**
116:     * Returns the value of field 'ECDataSourceExport'. The field
117:     * 'ECDataSourceExport' has the following description: Created
118:     * by Douglas Boyle, 2006-02-16
119:     *
120:     * @return the value of field 'ECDataSourceExport'.
121:     */
122:    public export.ECDataSourceExport getECDataSourceExport(
123:    ) {
124:        return this._ECDataSourceExport;
125:    }
126:
127:    /**
128:     *
129:     *
130:     * @param vECDataExport
131:     * @throws java.lang.IndexOutOfBoundsException if the index
132:     * given is outside the bounds of the collection
133:     */
134:    public void addECDataExport(
135:        final export.ECDataExport vECDataExport)
```

```
136:     throws java.lang.IndexOutOfBoundsException {
137:         this._ECDataExportList.add(vECDataExport);
138:     }
139:
140:     /**
141:      *
142:      *
143:      * @param index
144:      * @param vECDataExport
145:      * @throws java.lang.IndexOutOfBoundsException if the index
146:      * given is outside the bounds of the collection
147:      */
148:     public void addECDataExport(
149:         final int index,
150:         final export.Patient vECDataExport)
151:     throws java.lang.IndexOutOfBoundsException {
152:         this._ECDataExportList.add(index, vECDataExport);
153:     }
154:
155:     /**
156:      * Method enumerateECDataExport.
157:      *
158:      * @return an Enumeration over all possible elements of this
159:      * collection
160:      */
161:     public java.util Enumeration enumerateECDataExport(
162:     ) {
163:         return java.util.Collections.enumeration(this._ECDataExportList);
164:     }
165:
166:     /**
167:      * Method getPatient.
168:      *
169:      * @param index
170:      * @throws java.lang.IndexOutOfBoundsException if the index
171:      * given is outside the bounds of the collection
172:      * @return the value of the export.Patient at the given index
173:      */
174:     public export.ECDataExport getECDataExportList(
175:         final int index)
176:     throws java.lang.IndexOutOfBoundsException {
177:         // check bounds for index
178:         if (index < 0 || index >= this._ECDataExportList.size()) {
179:             throw new IndexOutOfBoundsException("getECDataExportList: Index value '" + index + "' not in range
[0.." + (this._ECDataExportList.size() - 1) + "]");
```



```
180:         }
181:
182:         return (export.ECDataExport) _ECDataExportList.get(index);
183:     }
184:
185:     /**
186:     * Method getPatient.Returns the contents of the collection in
187:     * an Array. <p>Note: Just in case the collection contents
188:     * are changing in another thread, we pass a 0-length Array of
189:     * the correct type into the API call. This way we <i>know</i>
190:     * that the Array returned is of exactly the correct length.
191:     *
192:     * @return this collection as an Array
193:     */
194:     public export.ECDataExport[] getECDataExport(
195:     ) {
196:         export.ECDataExport[] array = new export.ECDataExport[0];
197:         return (export.ECDataExport[]) this._ECDataExportList.toArray(array);
198:     }
199:
200:     /**
201:     * Method getECDataExportCount.
202:     *
203:     * @return the size of this collection
204:     */
205:     public int getECDataExportCount(
206:     ) {
207:         return this._ECDataExportList.size();
208:     }
209:
210:     /**
211:     * Method isValid.
212:     *
213:     * @return true if this object is valid according to the schema
214:     */
215:     public boolean isValid(
216:     ) {
217:         try {
218:             validate();
219:         } catch (org.exolab.castor.xml.ValidationException vex) {
220:             return false;
221:         }
222:         return true;
223:     }
224:
```

```
225:  /**
226:   * Method iterateECDataExport.
227:   *
228:   * @return an Iterator over all possible elements in this
229:   * collection
230:   */
231: public java.util.Iterator iterateECDataExport(
232: ) {
233:     return this._ECDataExportList.iterator();
234: }
235:
236: /**
237:  *
238:  *
239:  * @param out
240:  * @throws org.exolab.castor.xml.MarshalException if object is
241:  * null or if any SAXException is thrown during marshaling
242:  * @throws org.exolab.castor.xml.ValidationException if this
243:  * object is an invalid instance according to the schema
244:  */
245: public void marshal(
246:     final java.io.Writer out)
247: throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
248:     Marshaller.marshal(this, out);
249: }
250:
251: /**
252:  *
253:  *
254:  * @param handler
255:  * @throws java.io.IOException if an IOException occurs during
256:  * marshaling
257:  * @throws org.exolab.castor.xml.ValidationException if this
258:  * object is an invalid instance according to the schema
259:  * @throws org.exolab.castor.xml.MarshalException if object is
260:  * null or if any SAXException is thrown during marshaling
261:  */
262: public void marshal(
263:     final org.xml.sax.ContentHandler handler)
264: throws java.io.IOException, org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
265:     Marshaller.marshal(this, handler);
266: }
267:
268: /**
269:  */
```

```
270: public void removeAllECDataExport(
271: ) {
272:     this._ECDataExportList.clear();
273: }
274:
275: /**
276:  * Method removeECDataExport.
277:  *
278:  * @param vECDataExport
279:  * @return true if the object was removed from the collection.
280:  */
281: public boolean removeECDataExport(
282:     final export.ECDataExport vECDataExport) {
283:     boolean removed = _ECDataExportList.remove(vECDataExport);
284:     return removed;
285: }
286:
287: /**
288:  * Method removeECDataExport.
289:  *
290:  * @param index
291:  * @return the element removed from the collection
292:  */
293: public export.ECDataExport removeECDataExport(
294:     final int index) {
295:     java.lang.Object obj = this._ECDataExportList.remove(index);
296:     return (export.ECDataExport) obj;
297: }
298:
299: /**
300:  *
301:  *
302:  * @param index
303:  * @param vECDataExport
304:  * @throws java.lang.IndexOutOfBoundsException if the index
305:  * given is outside the bounds of the collection
306:  */
307: public void setECDataExport(
308:     final int index,
309:     final export.ECDataExport vECDataExport)
310:     throws java.lang.IndexOutOfBoundsException {
311:     // check bounds for index
312:     if (index < 0 || index >= this._ECDataExportList.size()) {
313:         throw new IndexOutOfBoundsException("setPatient: Index value '" + index + "' not in range [0.." +
(this._ECDataExportList.size() - 1) + "]);");
```

```
314:         }
315:
316:         this._ECDataExportList.set(index, vECDataExport);
317:     }
318:
319:     /**
320:      *
321:      *
322:      * @param vPatientAvECDataExportArrayrray
323:      */
324:     public void setECDataExport(
325:         final export.Patient[] vECDataExportArray) {
326:         //-- copy array
327:         _ECDataExportList.clear();
328:
329:         for (int i = 0; i < vECDataExportArray.length; i++) {
330:             this._ECDataExportList.add(vECDataExportArray[i]);
331:         }
332:     }
333:
334:     /**
335:      * Method unmarshal.
336:      *
337:      * @param reader
338:      * @throws org.exolab.castor.xml.MarshalException if object is
339:      * null or if any SAXException is thrown during marshaling
340:      * @throws org.exolab.castor.xml.ValidationException if this
341:      * object is an invalide instance according to the schema
342:      * @return the unmarshaled export.ECDataExport
343:      */
344:     public static export.ECDataExport unmarshal(
345:         final java.io.Reader reader)
346:     throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
347:         return (export.ECDataExport) Unmarshaller.unmarshal(export.ECDataExport.class, reader);
348:     }
349:
350:     /**
351:      *
352:      *
353:      * @throws org.exolab.castor.xml.ValidationException if this
354:      * object is an invalid instance according to the schema
355:      */
356:     public void validate(
357:     )
358:     throws org.exolab.castor.xml.ValidationException {
```

```
359:         org.exolab.castor.xml.Validator validator = new org.exolab.castor.xml.Validator();
360:         validator.validate(this);
361:     }
362:
363:     /** Aggiunto da Valentina
364:      * Method getECDataExportList (richiesto da Hibernate)
365:      */
366:     public java.util.List getECDataExportList(
367:     ) {
368:         return this._ECDataExportList;
369:     }
370:
371:     /** Aggiunto da Valentina
372:      * Method setPatientList uguale a setPatient (richiesto da Hibernate)
373:      */
374:     public void setECDataExportList(
375:         java.util.List eCDataExportList) {
376:         this._ECDataExportList= eCDataExportList;
377:     }
378:
379:
380:
381:
382: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      FieldExportProfiles.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: /**
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36:
37: package export;
38:
39:  //-----/
40:  //- Imported classes and packages -/
41:  //-----/
42:
43: import org.exolab.castor.xml.Marshaller;
44: import org.exolab.castor.xml.Unmarshaller;
45: import export.types.BIRODataSet;
```

```
46: import export.types.Ranking;
47:
48: /**
49:  * Class FieldExportProfiles.
50:  *
51:  * @version $Revision$ $Date$
52:  */
53: public class FieldExportProfiles implements java.io.Serializable {
54:
55:
56:     //-----/
57:     //- Class/Member Variables -/
58:     //-----/
59:     /**
60:     Aggiunto da Valentina
61:     */
62:     private int _ID = -1;
63:
64:     /**
65:     * Field _fieldName.
66:     */
67:     private BIRODataSet _fieldName;
68:
69:     /**
70:     * When were the field recording attributes below last reviewed?
71:     */
72:     private org.exolab.castor.types.Date _dateStatusLastReviewed;
73:
74:     /**
75:     * When were the field recording attributes below last reviewed?
76:     */
77:     private java.util.Date _hibernateDateStatusLastReviewed;
78:
79:     /**
80:     * Is the field recorded?
81:     */
82:     private boolean _recorded;
83:
84:     /**
85:     * keeps track of state for field: _recorded
86:     */
87:     private boolean _has_recorded;
88:
89:     /**
90:     * Is the field consistent with the BIRO definition?
```

```
91:      */
92:      private Ranking _consistency ;
93:
94:      /**
95:       * Is the field recording complete?
96:       */
97:      private long _completeness;
98:
99:      /**
100:       * keeps track of state for field: _completeness
101:       */
102:      private boolean _has_completeness;
103:
104:      /**
105:       * Is the field mandatory?
106:       */
107:      private boolean _mandatory;
108:
109:      /**
110:       * keeps track of state for field: _mandatory
111:       */
112:      private boolean _has_mandatory;
113:
114:      /**
115:       * Is the field routinely collected?
116:       */
117:      private boolean _routine;
118:
119:      /**
120:       * keeps track of state for field: _routine
121:       */
122:      private boolean _has_routine;
123:
124:      /**
125:       * Quality score as defined in Data Dictionary documentation
126:       * (Low/Medium/High)?
127:       */
128:      private Ranking _qualityScore;
129:
130:      /**
131:       * Field _fieldExportComments.
132:       */
133:      private java.lang.String _fieldExportComments;
134:
135:
```



```
136:         //-----/
137:         //- Constructors -/
138:         //-----/
139:
140:     public FieldExportProfiles() {
141:         super();
142:     }
143:
144:
145:         //-----/
146:         //- Methods -/
147:         //-----/
148:
149:     /**
150:      * @param vID
151:      */
152:     public void setID(
153:         final int vID)
154:     {
155:         this._ID = vID;
156:     }
157:
158:     /**
159:      * @param vID
160:      */
161:     public int getID()
162:     {
163:         return this._ID;
164:     }
165:
166:     /**
167:      */
168:     public void deleteCompleteness(
169:     ) {
170:         this._has_completeness= false;
171:     }
172:
173:     /**
174:      */
175:     public void deleteMandatory(
176:     ) {
177:         this._has_mandatory= false;
178:     }
179:
180:     /**
```

```
181:     */
182:     public void deleteRecorded(
183:     ) {
184:         this._has_recorded= false;
185:     }
186:
187:     /**
188:     */
189:     public void deleteRoutine(
190:     ) {
191:         this._has_routine= false;
192:     }
193:
194:     /**
195:     * Returns the value of field 'completeness'. The field
196:     * 'completeness' has the following description: Is the field
197:     * recording complete?
198:     *
199:     * @return the value of field 'Completeness'.
200:     */
201:     public long getCompleteness(
202:     ) {
203:         return this._completeness;
204:     }
205:
206:     /**
207:     * Returns the value of field 'consistency'. The field
208:     * 'consistency' has the following description: Is the field
209:     * consistent with the BIRO definition?
210:     *
211:     * @return the value of field 'Consistency'.
212:     */
213:     public Ranking getConsistency(
214:     ) {
215:         return this._consistency;
216:     }
217:
218:     /**
219:     * Returns the value of field 'dateStatusLastReviewed'. The
220:     * field 'dateStatusLastReviewed' has the following
221:     * description: When were the field recording attributes below
222:     * last reviewed?
223:     *
224:     * @return the value of field 'DateStatusLastReviewed'.
225:     */
```

```
226: public org.exolab.castor.types.Date getDateStatusLastReviewed(
227: ) {
228:     return this._dateStatusLastReviewed;
229: }
230:
231: /**
232:  * Returns the value of field 'hibernateDateStatusLastReviewed'. The
233:  * field 'hibernateDateStatusLastReviewed' has the following
234:  * description: When were the field recording attributes below
235:  * last reviewed?
236:  *
237:  * @return the value of field 'hibernateDateStatusLastReviewed'.
238:  */
239: public java.util.Date getHibernateDateStatusLastReviewed(
240: ) {
241:     this._hibernateDateStatusLastReviewed = this._dateStatusLastReviewed.toDate();
242:     return this._hibernateDateStatusLastReviewed;
243: }
244:
245: /**
246:  * Returns the value of field 'fieldExportComments'.
247:  *
248:  * @return the value of field 'FieldExportComments'.
249:  */
250: public java.lang.String getFieldExportComments(
251: ) {
252:     return this._fieldExportComments;
253: }
254:
255: /**
256:  * Returns the value of field 'fieldName'.
257:  *
258:  * @return the value of field 'FieldName'.
259:  */
260: public BIRODataSet getFieldName(
261: ) {
262:     return this._fieldName;
263: }
264:
265: /**
266:  * Returns the value of field 'mandatory'. The field
267:  * 'mandatory' has the following description: Is the field
268:  * mandatory?
269:  *
270:  * @return the value of field 'Mandatory'.
```

```
271:     */
272:     public boolean getMandatory(
273:     ) {
274:         return this._mandatory;
275:     }
276:
277:     /**
278:     * Returns the value of field 'qualityScore'. The field
279:     * 'qualityScore' has the following description: Quality score
280:     * as defined in Data Dictionary documentation
281:     * (Low/Medium/High)?
282:     *
283:     * @return the value of field 'QualityScore'.
284:     */
285:     public Ranking getQualityScore(
286:     ) {
287:         return this._qualityScore;
288:     }
289:
290:     /**
291:     * Returns the value of field 'recorded'. The field 'recorded'
292:     * has the following description: Is the field recorded?
293:     *
294:     * @return the value of field 'Recorded'.
295:     */
296:     public boolean getRecorded(
297:     ) {
298:         return this._recorded;
299:     }
300:
301:     /**
302:     * Returns the value of field 'routine'. The field 'routine'
303:     * has the following description: Is the field routinely
304:     * collected?
305:     *
306:     * @return the value of field 'Routine'.
307:     */
308:     public boolean getRoutine(
309:     ) {
310:         return this._routine;
311:     }
312:
313:     /**
314:     * Method hasCompleteness.
315:     *
316:     * @return true if at least one Completeness has been added
```

```
316:     */
317:     public boolean hasCompleteness(
318:     ) {
319:         return this._has_completeness;
320:     }
321:
322:     /**
323:      * Method hasMandatory.
324:      *
325:      * @return true if at least one Mandatory has been added
326:      */
327:     public boolean hasMandatory(
328:     ) {
329:         return this._has_mandatory;
330:     }
331:
332:     /**
333:      * Method hasRecorded.
334:      *
335:      * @return true if at least one Recorded has been added
336:      */
337:     public boolean hasRecorded(
338:     ) {
339:         return this._has_recorded;
340:     }
341:
342:     /**
343:      * Method hasRoutine.
344:      *
345:      * @return true if at least one Routine has been added
346:      */
347:     public boolean hasRoutine(
348:     ) {
349:         return this._has_routine;
350:     }
351:
352:     /**
353:      * Returns the value of field 'mandatory'. The field
354:      * 'mandatory' has the following description: Is the field
355:      * mandatory?
356:      *
357:      * @return the value of field 'Mandatory'.
358:      */
359:     public boolean isMandatory(
360:     ) {
```

```
361:         return this._mandatory;
362:     }
363:
364:     /**
365:     * Returns the value of field 'recorded'. The field 'recorded'
366:     * has the following description: Is the field recorded?
367:     *
368:     * @return the value of field 'Recorded'.
369:     */
370:     public boolean isRecorded(
371:     ) {
372:         return this._recorded;
373:     }
374:
375:     /**
376:     * Returns the value of field 'routine'. The field 'routine'
377:     * has the following description: Is the field routinely
378:     * collected?
379:     *
380:     * @return the value of field 'Routine'.
381:     */
382:     public boolean isRoutine(
383:     ) {
384:         return this._routine;
385:     }
386:
387:     /**
388:     * Method isValid.
389:     *
390:     * @return true if this object is valid according to the schema
391:     */
392:     public boolean isValid(
393:     ) {
394:         try {
395:             validate();
396:         } catch (org.exolab.castor.xml.ValidationException vex) {
397:             return false;
398:         }
399:         return true;
400:     }
401:
402:     /**
403:     *
404:     *
405:     * @param out
```

```
406:      * @throws org.exolab.castor.xml.MarshalException if object is
407:      * null or if any SAXException is thrown during marshaling
408:      * @throws org.exolab.castor.xml.ValidationException if this
409:      * object is an invalid instance according to the schema
410:      */
411:      public void marshal(
412:          final java.io.Writer out)
413:          throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
414:          Marshaller.marshal(this, out);
415:      }
416:
417:      /**
418:      *
419:      *
420:      * @param handler
421:      * @throws java.io.IOException if an IOException occurs during
422:      * marshaling
423:      * @throws org.exolab.castor.xml.ValidationException if this
424:      * object is an invalid instance according to the schema
425:      * @throws org.exolab.castor.xml.MarshalException if object is
426:      * null or if any SAXException is thrown during marshaling
427:      */
428:      public void marshal(
429:          final org.xml.sax.ContentHandler handler)
430:          throws java.io.IOException, org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
431:          Marshaller.marshal(this, handler);
432:      }
433:
434:      /**
435:      * Sets the value of field 'completeness'. The field
436:      * 'completeness' has the following description: Is the field
437:      * recording complete?
438:      *
439:      * @param completeness the value of field 'completeness'.
440:      */
441:      public void setCompleteness(
442:          final long completeness) {
443:          this._completeness = completeness;
444:          this._has_completeness = true;
445:      }
446:
447:      /**
448:      * Sets the value of field 'consistency'. The field
449:      * 'consistency' has the following description: Is the field
450:      * consistent with the BIRO definition?
```

```
451:      *
452:      * @param consistency the value of field 'consistency'.
453:      */
454:      public void setConsistency(
455:          final Ranking consistency) {
456:          this._consistency = consistency;
457:      }
458:
459:      /**
460:       * Sets the value of field 'dateStatusLastReviewed'. The field
461:       * 'dateStatusLastReviewed' has the following description: When
462:       * were the field recording attributes below last reviewed?
463:       *
464:       * @param dateStatusLastReviewed the value of field
465:       * 'dateStatusLastReviewed'.
466:       */
467:      public void setDateStatusLastReviewed(
468:          final org.exolab.castor.types.Date dateStatusLastReviewed) {
469:          this._dateStatusLastReviewed = dateStatusLastReviewed;
470:      }
471:
472:      /**
473:       * Sets the value of field 'HibernateDateStatusLastReviewed'. The field
474:       * 'hibernateDateStatusLastReviewed' has the following description: When
475:       * were the field recording attributes below last reviewed?
476:       *
477:       * @param hibernateDateStatusLastReviewed the value of field
478:       * 'dateStatusLastReviewed'.
479:       */
480:      public void setHibernateDateStatusLastReviewed(
481:          final java.util.Date hibernateDateStatusLastReviewed) {
482:          this._hibernateDateStatusLastReviewed = hibernateDateStatusLastReviewed;
483:      }
484:
485:      /**
486:       * Sets the value of field 'fieldExportComments'.
487:       *
488:       * @param fieldExportComments the value of field
489:       * 'fieldExportComments'.
490:       */
491:      public void setFieldExportComments(
492:          final java.lang.String fieldExportComments) {
493:          this._fieldExportComments = fieldExportComments;
494:      }
495:
```



```
496:  /**
497:   * Sets the value of field 'fieldName'.
498:   *
499:   * @param fieldName the value of field 'fieldName'.
500:   */
501:  public void setFieldName(
502:      final BIRODataSet fieldName) {
503:      this._fieldName = fieldName;
504:  }
505:
506:  /**
507:   * Sets the value of field 'mandatory'. The field 'mandatory'
508:   * has the following description: Is the field mandatory?
509:   *
510:   * @param mandatory the value of field 'mandatory'.
511:   */
512:  public void setMandatory(
513:      final boolean mandatory) {
514:      this._mandatory = mandatory;
515:      this._has_mandatory = true;
516:  }
517:
518:  /**
519:   * Sets the value of field 'qualityScore'. The field
520:   * 'qualityScore' has the following description: Quality score
521:   * as defined in Data Dictionary documentation
522:   * (Low/Medium/High)?
523:   *
524:   * @param qualityScore the value of field 'qualityScore'.
525:   */
526:  public void setQualityScore(
527:      final Ranking qualityScore) {
528:      this._qualityScore = qualityScore;
529:  }
530:
531:  /**
532:   * Sets the value of field 'recorded'. The field 'recorded' has
533:   * the following description: Is the field recorded?
534:   *
535:   * @param recorded the value of field 'recorded'.
536:   */
537:  public void setRecorded(
538:      final boolean recorded) {
539:      this._recorded = recorded;
540:      this._has_recorded = true;
```

```
541:     }
542:
543:     /**
544:      * Sets the value of field 'routine'. The field 'routine' has
545:      * the following description: Is the field routinely collected?
546:      *
547:      * @param routine the value of field 'routine'.
548:      */
549:     public void setRoutine(
550:         final boolean routine) {
551:         this._routine = routine;
552:         this._has_routine = true;
553:     }
554:
555:     /**
556:      * Method unmarshal.
557:      *
558:      * @param reader
559:      * @throws org.exolab.castor.xml.MarshalException if object is
560:      * null or if any SAXException is thrown during marshaling
561:      * @throws org.exolab.castor.xml.ValidationException if this
562:      * object is an invalid instance according to the schema
563:      * @return the unmarshaled DataSourceExport.FieldExportProfiles
564:      */
565:     public static export.FieldExportProfiles unmarshal(
566:         final java.io.Reader reader)
567:     throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
568:         return (export.FieldExportProfiles) Unmarshaller.unmarshal(export.FieldExportProfiles.class, reader);
569:     }
570:
571:     /**
572:      *
573:      *
574:      * @throws org.exolab.castor.xml.ValidationException if this
575:      * object is an invalid instance according to the schema
576:      */
577:     public void validate(
578:     )
579:     throws org.exolab.castor.xml.ValidationException {
580:         org.exolab.castor.xml.Validator validator = new org.exolab.castor.xml.Validator();
581:         validator.validate(this);
582:     }
583:
584: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      Patient.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: /**
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36: package export;
37:
38: //-----/
39: //- Imported classes and packages -/
40: //-----/
41: import org.exolab.castor.xml.Marshaller;
42: import org.exolab.castor.xml.Unmarshaller;
43:
44: /**
45:  * Class Patient.
```

```
46:  *
47:  * @version $Revision$ $Date$
48:  */
49: public class Patient implements java.io.Serializable {
50:
51:
52:     //-----/
53:     //- Class/Member Variables -/
54:     //-----/
55:     /**
56:     Aggiunto da Valentina
57:     */
58:     private int _ID = -1;
59:     /**
60:     * The patient profile is non-event-based data such as surname,
61:     * date of diagnosis and date of birth
62:     */
63:     private java.util.List _profileList;
64:     /**
65:     * Patients have events that happen chronologically (patient
66:     * episodes)
67:     */
68:     private java.util.List _episodeDataList;
69:     /**
70:     * Field _activityDataList.
71:     */
72:     private java.util.List _activityDataList;
73:
74:
75:     //-----/
76:     //- Constructors -/
77:     //-----/
78:     public Patient() {
79:         super();
80:         this._profileList = new java.util.ArrayList();
81:         this._episodeDataList = new java.util.ArrayList();
82:         this._activityDataList = new java.util.ArrayList();
83:     }
84:
85:
86:     //-----/
87:     //- Methods -/
88:     //-----/
89:     //aggiunto da Valentina
90:     /**
```

```
91:      * @param vID
92:      */
93:  public void setID(
94:      final int vID) {
95:      this._ID = vID;
96:  }
97:
98:  /**
99:   * @param vID
100:   */
101:  public int getID() {
102:      return this._ID;
103:  }
104:
105:  /**
106:   *
107:   *
108:   * @param vActivityData
109:   * @throws java.lang.IndexOutOfBoundsException if the index
110:   * given is outside the bounds of the collection
111:   */
112:  public void addActivityData(final ActivityData vActivityData)
113:      throws java.lang.IndexOutOfBoundsException {
114:      this._activityDataList.add(vActivityData);
115:  }
116:
117:  /**
118:   *
119:   *
120:   * @param index
121:   * @param vActivityData
122:   * @throws java.lang.IndexOutOfBoundsException if the index
123:   * given is outside the bounds of the collection
124:   */
125:  public void addActivityData(final int index, final ActivityData vActivityData)
126:      throws java.lang.IndexOutOfBoundsException {
127:      this._activityDataList.add(index, vActivityData);
128:  }
129:
130:  /**
131:   *
132:   *
133:   * @param vEpisodeData
134:   * @throws java.lang.IndexOutOfBoundsException if the index
135:   * given is outside the bounds of the collection
```

```
136:    */
137:    public void addEpisodeData(final export.EpisodeData vEpisodeData)
138:        throws java.lang.IndexOutOfBoundsException {
139:        this._episodeDataList.add(vEpisodeData);
140:    }
141:
142:    /**
143:     *
144:     *
145:     * @param index
146:     * @param vEpisodeData
147:     * @throws java.lang.IndexOutOfBoundsException if the index
148:     * given is outside the bounds of the collection
149:     */
150:    public void addEpisodeData(final int index, final export.EpisodeData vEpisodeData)
151:        throws java.lang.IndexOutOfBoundsException {
152:        this._episodeDataList.add(index, vEpisodeData);
153:    }
154:
155:    /**
156:     *
157:     *
158:     * @param vProfile
159:     * @throws java.lang.IndexOutOfBoundsException if the index
160:     * given is outside the bounds of the collection
161:     */
162:    public void addProfile(final export.Profile vProfile)
163:        throws java.lang.IndexOutOfBoundsException {
164:        this._profileList.add(vProfile);
165:    }
166:
167:    /**
168:     *
169:     *
170:     * @param index
171:     * @param vProfile
172:     * @throws java.lang.IndexOutOfBoundsException if the index
173:     * given is outside the bounds of the collection
174:     */
175:    public void addProfile(final int index, final export.Profile vProfile)
176:        throws java.lang.IndexOutOfBoundsException {
177:        this._profileList.add(index, vProfile);
178:    }
179:
180:    /**
```

```
181:      * Method enumerateActivityData.
182:      *
183:      * @return an Enumeration over all possible elements of this
184:      * collection
185:      */
186: public java.util.Enumeration enumerateActivityData() {
187:     return java.util.Collections.enumeration(this._activityDataList);
188: }
189:
190: /**
191:  * Method enumerateEpisodeData.
192:  *
193:  * @return an Enumeration over all possible elements of this
194:  * collection
195:  */
196: public java.util.Enumeration enumerateEpisodeData() {
197:     return java.util.Collections.enumeration(this._episodeDataList);
198: }
199:
200: /**
201:  * Method enumerateProfile.
202:  *
203:  * @return an Enumeration over all possible elements of this
204:  * collection
205:  */
206: public java.util.Enumeration enumerateProfile() {
207:     return java.util.Collections.enumeration(this._profileList);
208: }
209:
210: /**
211:  * Method getActivityData.
212:  *
213:  * @param index
214:  * @throws java.lang.IndexOutOfBoundsException if the index
215:  * given is outside the bounds of the collection
216:  * @return the value of the C:\Documents and
217:  * Settings\Valentina\Desktop\schema\classes.ActivityData at
218:  * the given index
219:  */
220: public ActivityData getActivityData(final int index)
221:     throws java.lang.IndexOutOfBoundsException {
222:     // check bounds for index
223:     if (index < 0 || index >= this._activityDataList.size()) {
224:         throw new IndexOutOfBoundsException("getActivityData: Index value '" + index + "' not in range [0.." +
(this._activityDataList.size() - 1) + "]);");
```

```
225:         }
226:
227:         return (ActivityData) _activityDataList.get(index);
228:     }
229:
230:     /**
231:      * Method getActivityData.Returns the contents of the
232:      * collection in an Array. <p>Note: Just in case the
233:      * collection contents are changing in another thread, we pass
234:      * a 0-length Array of the correct type into the API call.
235:      * This way we <i>know</i> that the Array returned is of
236:      * exactly the correct length.
237:      *
238:      * @return this collection as an Array
239:      */
240:     public ActivityData[] getActivityData() {
241:         ActivityData[] array = new ActivityData[0];
242:         return (ActivityData[]) this._activityDataList.toArray(array);
243:     }
244:
245:     /**
246:      * Method getActivityDataCount.
247:      *
248:      * @return the size of this collection
249:      */
250:     public int getActivityDataCount() {
251:         return this._activityDataList.size();
252:     }
253:
254:     /**
255:      * Method getEpisodeData.
256:      *
257:      * @param index
258:      * @throws java.lang.IndexOutOfBoundsException if the index
259:      * given is outside the bounds of the collection
260:      * @return the value of the dataExport.EpisodeData at the given
261:      * index
262:      */
263:     public export.EpisodeData getEpisodeData(final int index)
264:         throws java.lang.IndexOutOfBoundsException {
265:         // check bounds for index
266:         if (index < 0 || index >= this._episodeDataList.size()) {
267:             throw new IndexOutOfBoundsException("getEpisodeData: Index value '" + index + "' not in range [0.." +
268: (this._episodeDataList.size() - 1) + "];");
269:         }
```



```
269:
270:         return (export.EpisodeData) _episodeDataList.get(index);
271:     }
272:
273:     /**
274:     * Method getEpisodeData.Returns the contents of the collection
275:     * in an Array. <p>Note: Just in case the collection contents
276:     * are changing in another thread, we pass a 0-length Array of
277:     * the correct type into the API call. This way we <i>know</i>
278:     * that the Array returned is of exactly the correct length.
279:     *
280:     * @return this collection as an Array
281:     */
282:     public export.EpisodeData[] getEpisodeData() {
283:         export.EpisodeData[] array = new export.EpisodeData[0];
284:         return (export.EpisodeData[]) this._episodeDataList.toArray(array);
285:     }
286:
287:     /**
288:     * Method getEpisodeDataCount.
289:     *
290:     * @return the size of this collection
291:     */
292:     public int getEpisodeDataCount() {
293:         return this._episodeDataList.size();
294:     }
295:
296:     /**
297:     * Method getProfile.
298:     *
299:     * @param index
300:     * @throws java.lang.IndexOutOfBoundsException if the index
301:     * given is outside the bounds of the collection
302:     * @return the value of the dataExport.Profile at the given inde
303:     */
304:     public export.Profile getProfile(final int index)
305:         throws java.lang.IndexOutOfBoundsException {
306:         // check bounds for index
307:         if (index < 0 || index >= this._profileList.size()) {
308:             throw new IndexOutOfBoundsException("getProfile: Index value '" + index + "' not in range [0.." +
309: (this._profileList.size() - 1) + "]");
310:         }
311:         return (export.Profile) _profileList.get(index);
312:     }
```

```
313:
314: /**
315:  * Method getProfile.Returns the contents of the collection in
316:  * an Array. <p>Note: Just in case the collection contents
317:  * are changing in another thread, we pass a 0-length Array of
318:  * the correct type into the API call. This way we <i>know</i>
319:  * that the Array returned is of exactly the correct length.
320:  *
321:  * @return this collection as an Array
322:  */
323: public export.Profile[] getProfile() {
324:     export.Profile[] array = new export.Profile[0];
325:     return (export.Profile[]) this._profileList.toArray(array);
326: }
327:
328: /**
329:  * Method getProfileCount.
330:  *
331:  * @return the size of this collection
332:  */
333: public int getProfileCount() {
334:     return this._profileList.size();
335: }
336:
337: /**
338:  * Method isValid.
339:  *
340:  * @return true if this object is valid according to the schema
341:  */
342: public boolean isValid() {
343:     try {
344:         validate();
345:     } catch (org.exolab.castor.xml.ValidationException vex) {
346:         return false;
347:     }
348:     return true;
349: }
350:
351: /**
352:  * Method iterateActivityData.
353:  *
354:  * @return an Iterator over all possible elements in this
355:  * collection
356:  */
357: public java.util.Iterator iterateActivityData() {
```

```
358:         return this._activityDataList.iterator();
359:     }
360:
361:     /**
362:      * Method iterateEpisodeData.
363:      *
364:      * @return an Iterator over all possible elements in this
365:      * collection
366:      */
367:     public java.util.Iterator iterateEpisodeData() {
368:         return this._episodeDataList.iterator();
369:     }
370:
371:     /**
372:      * Method iterateProfile.
373:      *
374:      * @return an Iterator over all possible elements in this
375:      * collection
376:      */
377:     public java.util.Iterator iterateProfile() {
378:         return this._profileList.iterator();
379:     }
380:
381:     /**
382:      *
383:      *
384:      * @param out
385:      * @throws org.exolab.castor.xml.MarshalException if object is
386:      * null or if any SAXException is thrown during marshaling
387:      * @throws org.exolab.castor.xml.ValidationException if this
388:      * object is an invalid instance according to the schema
389:      */
390:     public void marshal(
391:         final java.io.Writer out)
392:         throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
393:         Marshaller.marshal(this, out);
394:     }
395:
396:     /**
397:      *
398:      *
399:      * @param handler
400:      * @throws java.io.IOException if an IOException occurs during
401:      * marshaling
402:      * @throws org.exolab.castor.xml.ValidationException if this
```

```
403:      * object is an invalid instance according to the schema
404:      * @throws org.exolab.castor.xml.MarshalException if object is
405:      * null or if any SAXException is thrown during marshaling
406:      */
407:      public void marshal(
408:          final org.xml.sax.ContentHandler handler)
409:          throws java.io.IOException, org.exolab.castor.xml.MarshalException,
org.exolab.castor.xml.ValidationException {
410:          Marshaller.marshal(this, handler);
411:      }
412:  /**
413:      * Method removeActivityData.
414:      *
415:      * @param vActivityData
416:      * @return true if the object was removed from the collection.
417:      */
418:      public boolean removeActivityData(
419:          final ActivityData vActivityData) {
420:          boolean removed = _activityDataList.remove(vActivityData);
421:          return removed;
422:      }
423:
424:  /**
425:      * Method removeActivityDataAt.
426:      *
427:      * @param index
428:      * @return the element removed from the collection
429:      */
430:      public ActivityData removeActivityDataAt(
431:          final int index) {
432:          java.lang.Object obj = this._activityDataList.remove(index);
433:          return (ActivityData) obj;
434:      }
435:
436:  /**
437:      */
438:      public void removeAllActivityData() {
439:          this._activityDataList.clear();
440:      }
441:
442:  /**
443:      */
444:      public void removeAllEpisodeData() {
445:          this._episodeDataList.clear();
446:      }
```

```
447:
448: /**
449:  */
450: public void removeAllProfile() {
451:     this._profileList.clear();
452: }
453:
454: /**
455:  * Method removeEpisodeData.
456:  *
457:  * @param vEpisodeData
458:  * @return true if the object was removed from the collection.
459:  */
460: public boolean removeEpisodeData(final export.EpisodeData vEpisodeData) {
461:     boolean removed = _episodeDataList.remove(vEpisodeData);
462:     return removed;
463: }
464:
465: /**
466:  * Method removeEpisodeDataAt.
467:  *
468:  * @param index
469:  * @return the element removed from the collection
470:  */
471: public export.EpisodeData removeEpisodeDataAt(
472:     final int index) {
473:     java.lang.Object obj = this._episodeDataList.remove(index);
474:     return (export.EpisodeData) obj;
475: }
476:
477: /**
478:  * Method removeProfile.
479:  *
480:  * @param vProfile
481:  * @return true if the object was removed from the collection.
482:  */
483: public boolean removeProfile(final export.Profile vProfile) {
484:     boolean removed = _profileList.remove(vProfile);
485:     return removed;
486: }
487:
488: /**
489:  * Method removeProfileAt.
490:  *
491:  * @param index
```

```
492:      * @return the element removed from the collection
493:      */
494:      public export.Profile removeProfileAt(final int index) {
495:          java.lang.Object obj = this._profileList.remove(index);
496:          return (export.Profile) obj;
497:      }
498:  /**
499:   *
500:   *
501:   * @param index
502:   * @param vActivityData
503:   * @throws java.lang.IndexOutOfBoundsException if the index
504:   * given is outside the bounds of the collection
505:   */
506:   public void setActivityData(
507:       final int index,
508:       final ActivityData vActivityData)
509:       throws java.lang.IndexOutOfBoundsException {
510:       // check bounds for index
511:       if (index < 0 || index >= this._activityDataList.size()) {
512:           throw new IndexOutOfBoundsException("setActivityData: Index value '" + index + "' not in range [0.." +
(this._activityDataList.size() - 1) + "]");
513:       }
514:
515:       this._activityDataList.set(index, vActivityData);
516:   }
517:
518:  /**
519:   *
520:   *
521:   * @param vActivityDataArray
522:   */
523:   public void setActivityData(
524:       final ActivityData[] vActivityDataArray) {
525:       //-- copy array
526:       _activityDataList.clear();
527:
528:       for (int i = 0; i < vActivityDataArray.length; i++) {
529:           this._activityDataList.add(vActivityDataArray[i]);
530:       }
531:   }
532:
533:  /**
534:   *
535:   *
```

```
536:      * @param index
537:      * @param vEpisodeData
538:      * @throws java.lang.IndexOutOfBoundsException if the index
539:      * given is outside the bounds of the collection
540:      */
541:      public void setEpisodeData(final int index, final export.EpisodeData vEpisodeData)
542:          throws java.lang.IndexOutOfBoundsException {
543:          // check bounds for index
544:          if (index < 0 || index >= this._episodeDataList.size()) {
545:              throw new IndexOutOfBoundsException("setEpisodeData: Index value '" + index + "' not in range [0.." +
(this._episodeDataList.size() - 1) + "]");
546:          }
547:
548:          this._episodeDataList.set(index, vEpisodeData);
549:      }
550:
551:      /**
552:       *
553:       *
554:       * @param vEpisodeDataArray
555:       */
556:      public void setEpisodeData(final export.EpisodeData[] vEpisodeDataArray) {
557:          //-- copy array
558:          _episodeDataList.clear();
559:
560:          for (int i = 0; i < vEpisodeDataArray.length; i++) {
561:              this._episodeDataList.add(vEpisodeDataArray[i]);
562:          }
563:      }
564:
565:      /**
566:       *
567:       *
568:       * @param index
569:       * @param vProfile
570:       * @throws java.lang.IndexOutOfBoundsException if the index
571:       * given is outside the bounds of the collection
572:       */
573:      public void setProfile(final int index, final export.Profile vProfile)
574:          throws java.lang.IndexOutOfBoundsException {
575:          // check bounds for index
576:          if (index < 0 || index >= this._profileList.size()) {
577:              throw new IndexOutOfBoundsException("setProfile: Index value '" + index + "' not in range [0.." +
(this._profileList.size() - 1) + "]");
578:          }
```

```
579:
580:     this._profileList.set(index, vProfile);
581: }
582:
583: /**
584:  *
585:  *
586:  * @param vProfileArray
587:  */
588: public void setProfile(final export.Profile[] vProfileArray) {
589:     //-- copy array
590:     _profileList.clear();
591:
592:     for (int i = 0; i < vProfileArray.length; i++) {
593:         this._profileList.add(vProfileArray[i]);
594:     }
595: }
596:
597: /**
598:  * Method unmarshal.
599:  *
600:  * @param reader
601:  * @throws org.exolab.castor.xml.MarshalException if object is
602:  * null or if any SAXException is thrown during marshaling
603:  * @throws org.exolab.castor.xml.ValidationException if this
604:  * object is an invalid instance according to the schema
605:  * @return the unmarshaled dataExport.Patient
606:  */
607: public static export.Patient unmarshal(final java.io.Reader reader)
608:     throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
609:     return (export.Patient) Unmarshaller.unmarshal(export.Patient.class, reader);
610: }
611:
612: /**
613:  *
614:  *
615:  * @throws org.exolab.castor.xml.ValidationException if this
616:  * object is an invalid instance according to the schema
617:  */
618: public void validate()
619:     throws org.exolab.castor.xml.ValidationException {
620:     org.exolab.castor.xml.Validator validator = new org.exolab.castor.xml.Validator();
621:     validator.validate(this);
622: }
623: /** Aggiunto da Valentina
```



```
624:      * Method getProfileList (richiesto da Hibernate)
625:      */
626:  public java.util.List getProfileList() {
627:      return this._profileList;
628:  }
629:
630:  /** Aggiunto da Valentina
631:   * Method setProfileList uguale a setProfile (richiesto da Hibernate)
632:   */
633:  public void setProfileList(
634:      java.util.List profileList) {
635:      this._profileList = profileList;
636:  }
637:
638:  /** Aggiunto da Valentina
639:   * Method getActivityDataList (richiesto da Hibernate)
640:   */
641:  public java.util.List getActivityDataList() {
642:      return this._activityDataList;
643:  }
644:
645:  /** Aggiunto da Valentina
646:   * Method setActivityDataList uguale a setActivityData (richiesto da Hibernate)
647:   */
648:  public void setActivityDataList(java.util.List activityDataList) {
649:      this._activityDataList = activityDataList;
650:  }
651:  }
652:
653:  /** Aggiunto da Valentina
654:   * Method getEpisodeDataList (richiesto da Hibernate)
655:   */
656:  public java.util.List getEpisodeDataList() {
657:      return this._episodeDataList;
658:  }
659:
660:  /** Aggiunto da Valentina
661:   * Method setEpisodeDataList uguale a setEpisodeData (richiesto da Hibernate)
662:   */
663:  public void setEpisodeDataList(java.util.List episodeDataList) {
664:      this._episodeDataList = episodeDataList;
665:  }
666:  }
667: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      Profile.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: /**
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36:
37: package export;
38:
39:  //---------------------------------/
40:  //- Imported classes and packages -/
41:  //---------------------------------/
42:
43: import export.types.BIRODataSet;
44: import org.exolab.castor.xml.Marshaller;
45: import org.exolab.castor.xml.Unmarshaller;
```

```
46:
47: /**
48:  * The patient profile is non-event-based data such as surname,
49:  * date of diagnosis and date of birth
50:  *
51:  * @version $Revision$ $Date$
52:  */
53: public class Profile implements java.io.Serializable {
54:
55:
56:     //-----/
57:     //- Class/Member Variables -/
58:     //-----/
59:     /**
60:     Aggiunto da Valentina
61:     */
62:     private int _ID    =-1;
63:
64:     /**
65:     * Standard BIRO field name
66:     */
67:     private BIRODataSet _profileFieldName;
68:
69:     /**
70:     * Field result
71:     */
72:     private java.lang.String _profileFieldValue;
73:
74:
75:     //-----/
76:     //- Constructors -/
77:     //-----/
78:
79:     public Profile() {
80:         super();
81:     }
82:
83:
84:     //-----/
85:     //- Methods -/
86:     //-----/
87:     //aggiunto da Valentina
88:     /**
89:     * @param vID
90:     */
```

```
91: public void setID(
92:     final int vID)
93: {
94:     this._ID = vID;
95: }
96:
97: /**
98:  * @param vID
99:  */
100: public int getID()
101: {
102:     return this._ID;
103: }
104:
105: /**
106:  * Returns the value of field 'profileFieldName'. The field
107:  * 'profileFieldName' has the following description: Standard
108:  * BIRO field name
109:  *
110:  * @return the value of field 'ProfileFieldName'.
111:  */
112: public BIRODataSet getProfileFieldName(
113: ) {
114:     return this._profileFieldName;
115: }
116:
117: /**
118:  * Returns the value of field 'profileFieldValue'. The field
119:  * 'profileFieldValue' has the following description: Field
120:  * result
121:  *
122:  * @return the value of field 'ProfileFieldValue'.
123:  */
124: public java.lang.String getProfileFieldValue(
125: ) {
126:     return this._profileFieldValue;
127: }
128:
129: /**
130:  * Method isValid.
131:  *
132:  * @return true if this object is valid according to the schema
133:  */
134: public boolean isValid(
135: ) {
```

```
136:         try {
137:             validate();
138:         } catch (org.exolab.castor.xml.ValidationException vex) {
139:             return false;
140:         }
141:         return true;
142:     }
143:
144: /**
145:  *
146:  *
147:  * @param out
148:  * @throws org.exolab.castor.xml.MarshalException if object is
149:  * null or if any SAXException is thrown during marshaling
150:  * @throws org.exolab.castor.xml.ValidationException if this
151:  * object is an invalid instance according to the schema
152:  */
153: public void marshal(
154:     final java.io.Writer out)
155: throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
156:     Marshaller.marshal(this, out);
157: }
158:
159: /**
160:  *
161:  *
162:  * @param handler
163:  * @throws java.io.IOException if an IOException occurs during
164:  * marshaling
165:  * @throws org.exolab.castor.xml.ValidationException if this
166:  * object is an invalid instance according to the schema
167:  * @throws org.exolab.castor.xml.MarshalException if object is
168:  * null or if any SAXException is thrown during marshaling
169:  */
170: public void marshal(
171:     final org.xml.sax.ContentHandler handler)
172: throws java.io.IOException, org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
173:     Marshaller.marshal(this, handler);
174: }
175:
176: /**
177:  * Sets the value of field 'profileFieldName'. The field
178:  * 'profileFieldName' has the following description: Standard
179:  * BIRO field name
180:  *
```

```
181:      * @param profileFieldName the value of field 'profileFieldName'
182:      */
183:      public void setProfileFieldName(
184:          final BIRODataSet profileFieldName) {
185:          this._profileFieldName = profileFieldName;
186:      }
187:
188:      /**
189:       * Sets the value of field 'profileFieldValue'. The field
190:       * 'profileFieldValue' has the following description: Field
191:       * result
192:       *
193:       * @param profileFieldValue the value of field
194:       * 'profileFieldValue'.
195:       */
196:      public void setProfileFieldValue(
197:          final java.lang.String profileFieldValue) {
198:          this._profileFieldValue = profileFieldValue;
199:      }
200:
201:      /**
202:       * Method unmarshal.
203:       *
204:       * @param reader
205:       * @throws org.exolab.castor.xml.MarshalException if object is
206:       * null or if any SAXException is thrown during marshaling
207:       * @throws org.exolab.castor.xml.ValidationException if this
208:       * object is an invalid instance according to the schema
209:       * @return the unmarshaled dataExport.Profile
210:       */
211:      public static export.Profile unmarshal(
212:          final java.io.Reader reader)
213:          throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
214:          return (export.Profile) Unmarshaller.unmarshal(export.Profile.class, reader);
215:      }
216:
217:      /**
218:       *
219:       *
220:       * @throws org.exolab.castor.xml.ValidationException if this
221:       * object is an invalid instance according to the schema
222:       */
223:      public void validate(
224:      )
225:      throws org.exolab.castor.xml.ValidationException {
```

```
226:         org.exolab.castor.xml.Validator validator = new org.exolab.castor.xml.Validator();
227:         validator.validate(this);
228:     }
229:
230: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      SiteHeader.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  **/
30: /**
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36: package export;
37:
38: //-----/
39: //- Imported classes and packages -/
40: //-----/
41: import org.exolab.castor.xml.Marshaller;
42: import org.exolab.castor.xml.Unmarshaller;
43: import export.types.DataSource;
44: /**
45:  * Class SiteHeader.
```



```
46:  *
47:  * @version $Revision$ $Date$
48:  */
49: public class SiteHeader implements java.io.Serializable {
50:
51:
52:     //-----/
53:     //- Class/Member Variables -/
54:     //-----/
55:     /**
56:     Aggiunto da Valentina
57:     */
58:     private int _ID =-1;
59:     /**
60:     Aggiunto da Valentina
61:     */
62:     private java.lang.String _DS_NAME;
63:     /**
64:     * Field _dateHeaderInformationChecked.
65:     */
66:     private org.exolab.castor.types.Date _dateHeaderInformationChecked;
67:     /**
68:     * Field _hibernateDateHeaderInformationChecked.
69:     */
70:     private java.util.Date _hibernateDateHeaderInformationChecked;
71:     /**
72:     * Unique centre identification number (Defined as a BIRO
73:     * Clinical Site)
74:     */
75:     private DataSource _DS_ID;
76:     /**
77:     * Internet address for Data Source
78:     */
79:     private java.lang.String _DS_WEBSITE;
80:     /**
81:     * First line of Data Source address
82:     */
83:     private java.lang.String _DS_ADDRESS_1;
84:     /**
85:     * Second line of Data Source address
86:     */
87:     private java.lang.String _DS_ADDRESS_2;
88:     /**
89:     * Third line of Data Source address
90:     */
```

```
91: private java.lang.String _DS_ADDRESS_3;
92: /**
93:  * Fourth line of Data Source address
94:  */
95: private java.lang.String _DS_ADDRESS_4;
96: /**
97:  * Post Code of Data Source
98:  */
99: private java.lang.String _DS_POST_CODE;
100: /**
101:  * The country from which the clinical data originates
102:  */
103: private java.lang.String _DS_COUNTRY;
104: /**
105:  * Clinical representative from Data Source
106:  */
107: private java.lang.String _DS_C_CONTACT;
108: /**
109:  * Email address of Data Source clinical representative
110:  */
111: private java.lang.String _DS_C_EMAIL;
112: /**
113:  * Technical representative from Data Source
114:  */
115: private java.lang.String _DS_T_CONTACT;
116: /**
117:  * Email address of Data Source technical representative
118:  */
119: private java.lang.String _DS_T_EMAIL;
120: /**
121:  * Field _headerComments.
122:  */
123: private java.lang.String _headerComments;
124:
125:
126: //-----/
127: //- Constructors -/
128: //-----/
129: public SiteHeader() {
130:     super();
131: }
132:
133:
134: //-----/
135: //- Methods -/
```

```
136: //-----/
137: /**
138:  * @param vID
139:  */
140: public void setID(
141:     final int vID) {
142:     this._ID = vID;
143: }
144:
145: /**
146:  * @param vID
147:  */
148: public int getID() {
149:     return this._ID;
150: }
151:
152: public String getDS_NAME() {
153:     return _DS_NAME;
154: }
155:
156: public void setDS_NAME(String DS_NAME) {
157:     this._DS_NAME = DS_NAME;
158: }
159:
160: /**
161:  * Returns the value of field 'DS_ADDRESS_1'. The field
162:  * 'DS_ADDRESS_1' has the following description: First line of
163:  * Data Source address
164:  *
165:  * @return the value of field 'DS_ADDRESS_1'.
166:  */
167: public java.lang.String getDS_ADDRESS_1() {
168:     return this._DS_ADDRESS_1;
169: }
170:
171: /**
172:  * Returns the value of field 'DS_ADDRESS_2'. The field
173:  * 'DS_ADDRESS_2' has the following description: Second line of
174:  * Data Source address
175:  *
176:  * @return the value of field 'DS_ADDRESS_2'.
177:  */
178: public java.lang.String getDS_ADDRESS_2() {
179:     return this._DS_ADDRESS_2;
180: }
```

```
181:
182: /**
183:  * Returns the value of field 'DS_ADDRESS_3'. The field
184:  * 'DS_ADDRESS_3' has the following description: Third line of
185:  * Data Source address
186:  *
187:  * @return the value of field 'DS_ADDRESS_3'.
188:  */
189: public java.lang.String getDS_ADDRESS_3() {
190:     return this._DS_ADDRESS_3;
191: }
192:
193: /**
194:  * Returns the value of field 'DS_ADDRESS_4'. The field
195:  * 'DS_ADDRESS_4' has the following description: Fourth line of
196:  * Data Source address
197:  *
198:  * @return the value of field 'DS_ADDRESS_4'.
199:  */
200: public java.lang.String getDS_ADDRESS_4() {
201:     return this._DS_ADDRESS_4;
202: }
203:
204: /**
205:  * Returns the value of field 'DS_COUNTRY'. The field
206:  * 'DS_COUNTRY' has the following description: The country from
207:  * which the clinical data originates
208:  *
209:  * @return the value of field 'DS_COUNTRY'.
210:  */
211: public java.lang.String getDS_COUNTRY() {
212:     return this._DS_COUNTRY;
213: }
214:
215: /**
216:  * Returns the value of field 'DS_C_CONTACT'. The field
217:  * 'DS_C_CONTACT' has the following description: Clinical
218:  * representative from Data Source
219:  *
220:  * @return the value of field 'DS_C_CONTACT'.
221:  */
222: public java.lang.String getDS_C_CONTACT() {
223:     return this._DS_C_CONTACT;
224: }
225:
```

```
226:  /**
227:   * Returns the value of field 'DS_C_EMAIL'. The field
228:   * 'DS_C_EMAIL' has the following description: Email address of
229:   * Data Source clinical representative
230:   *
231:   * @return the value of field 'DS_C_EMAIL'.
232:   */
233:  public java.lang.String getDS_C_EMAIL() {
234:      return this._DS_C_EMAIL;
235:  }
236:
237:  /**
238:   * Returns the value of field 'DS_ID'. The field 'DS_ID' has
239:   * the following description: Unique centre identification
240:   * number (Defined as a BIRO Clinical Site)
241:   *
242:   * @return the value of field 'DS_ID'.
243:   */
244:  public DataSource getDS_ID() {
245:      return this._DS_ID;
246:  }
247:
248:  /**
249:   * Returns the value of field 'DS_POST_CODE'. The field
250:   * 'DS_POST_CODE' has the following description: Post Code of
251:   * Data Source
252:   *
253:   * @return the value of field 'DS_POST_CODE'.
254:   */
255:  public java.lang.String getDS_POST_CODE() {
256:      return this._DS_POST_CODE;
257:  }
258:
259:  /**
260:   * Returns the value of field 'DS_T_CONTACT'. The field
261:   * 'DS_T_CONTACT' has the following description: Technical
262:   * representative from Data Source
263:   *
264:   * @return the value of field 'DS_T_CONTACT'.
265:   */
266:  public java.lang.String getDS_T_CONTACT() {
267:      return this._DS_T_CONTACT;
268:  }
269:
270:  /**
```

```
271:      * Returns the value of field 'DS_T_EMAIL'. The field
272:      * 'DS_T_EMAIL' has the following description: Email address of
273:      * Data Source technical representative
274:      *
275:      * @return the value of field 'DS_T_EMAIL'.
276:      */
277: public java.lang.String getDS_T_EMAIL() {
278:     return this._DS_T_EMAIL;
279: }
280:
281: /**
282:  * Returns the value of field 'DS_WEBSITE'. The field
283:  * 'DS_WEBSITE' has the following description: Internet address
284:  * for Data Source
285:  *
286:  * @return the value of field 'DS_WEBSITE'.
287:  */
288: public java.lang.String getDS_WEBSITE() {
289:     return this._DS_WEBSITE;
290: }
291:
292: /**
293:  * Returns the value of field 'dateHeaderInformationChecked'.
294:  *
295:  * @return the value of field 'DateHeaderInformationChecked'.
296:  */
297: public org.exolab.castor.types.Date getDateHeaderInformationChecked() {
298:     return this._dateHeaderInformationChecked;
299: }
300:
301: /**
302:  * Returns the value of field 'hibernateDateHeaderInformationChecked'.
303:  *
304:  * @return the value of field 'hibernateDateHeaderInformationChecked'.
305:  */
306: public java.util.Date getHibernateDateHeaderInformationChecked() {
307:     this._hibernateDateHeaderInformationChecked = this._dateHeaderInformationChecked.toDate();
308:     return this._hibernateDateHeaderInformationChecked;
309: }
310:
311: /**
312:  * Returns the value of field 'headerComments'.
313:  *
314:  * @return the value of field 'HeaderComments'.
315:  */
```

```
316: public java.lang.String getHeaderComments() {
317:     return this._headerComments;
318: }
319:
320: /**
321:  * Method isValid.
322:  *
323:  * @return true if this object is valid according to the schema
324:  */
325: public boolean isValid() {
326:     try {
327:         validate();
328:     } catch (org.exolab.castor.xml.ValidationException vex) {
329:         return false;
330:     }
331:     return true;
332: }
333:
334: /**
335:  *
336:  *
337:  * @param out
338:  * @throws org.exolab.castor.xml.MarshalException if object is
339:  * null or if any SAXException is thrown during marshaling
340:  * @throws org.exolab.castor.xml.ValidationException if this
341:  * object is an invalid instance according to the schema
342:  */
343: public void marshal(
344:     final java.io.Writer out)
345:     throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
346:     Marshaller.marshal(this, out);
347: }
348:
349: /**
350:  *
351:  *
352:  * @param handler
353:  * @throws java.io.IOException if an IOException occurs during
354:  * marshaling
355:  * @throws org.exolab.castor.xml.ValidationException if this
356:  * object is an invalid instance according to the schema
357:  * @throws org.exolab.castor.xml.MarshalException if object is
358:  * null or if any SAXException is thrown during marshaling
359:  */
360: public void marshal(
```

```
361:         final org.xml.sax.ContentHandler handler)
362:         throws java.io.IOException, org.exolab.castor.xml.MarshalException,
org.exolab.castor.xml.ValidationException {
363:     Marshaller.marshall(this, handler);
364: }
365:
366: /**
367:  * Sets the value of field 'DS_ADDRESS_1'. The field
368:  * 'DS_ADDRESS_1' has the following description: First line of
369:  * Data Source address
370:  *
371:  * @param DS_ADDRESS_1 the value of field 'DS_ADDRESS_1'.
372:  */
373: public void setDS_ADDRESS_1(
374:     final java.lang.String DS_ADDRESS_1) {
375:     this._DS_ADDRESS_1 = DS_ADDRESS_1;
376: }
377:
378: /**
379:  * Sets the value of field 'DS_ADDRESS_2'. The field
380:  * 'DS_ADDRESS_2' has the following description: Second line of
381:  * Data Source address
382:  *
383:  * @param DS_ADDRESS_2 the value of field 'DS_ADDRESS_2'.
384:  */
385: public void setDS_ADDRESS_2(
386:     final java.lang.String DS_ADDRESS_2) {
387:     this._DS_ADDRESS_2 = DS_ADDRESS_2;
388: }
389:
390: /**
391:  * Sets the value of field 'DS_ADDRESS_3'. The field
392:  * 'DS_ADDRESS_3' has the following description: Third line of
393:  * Data Source address
394:  *
395:  * @param DS_ADDRESS_3 the value of field 'DS_ADDRESS_3'.
396:  */
397: public void setDS_ADDRESS_3(
398:     final java.lang.String DS_ADDRESS_3) {
399:     this._DS_ADDRESS_3 = DS_ADDRESS_3;
400: }
401:
402: /**
403:  * Sets the value of field 'DS_ADDRESS_4'. The field
404:  * 'DS_ADDRESS_4' has the following description: Fourth line of
```



```
405:      * Data Source address
406:      *
407:      * @param DS_ADDRESS_4 the value of field 'DS_ADDRESS_4'.
408:      */
409:  public void setDS_ADDRESS_4(
410:      final java.lang.String DS_ADDRESS_4) {
411:      this._DS_ADDRESS_4 = DS_ADDRESS_4;
412:  }
413:
414:  /**
415:   * Sets the value of field 'DS_COUNTRY'. The field 'DS_COUNTRY'
416:   * has the following description: The country from which the
417:   * clinical data originates
418:   *
419:   * @param DS_COUNTRY the value of field 'DS_COUNTRY'.
420:   */
421:  public void setDS_COUNTRY(
422:      final java.lang.String DS_COUNTRY) {
423:      this._DS_COUNTRY = DS_COUNTRY;
424:  }
425:
426:  /**
427:   * Sets the value of field 'DS_C_CONTACT'. The field
428:   * 'DS_C_CONTACT' has the following description: Clinical
429:   * representative from Data Source
430:   *
431:   * @param DS_C_CONTACT the value of field 'DS_C_CONTACT'.
432:   */
433:  public void setDS_C_CONTACT(
434:      final java.lang.String DS_C_CONTACT) {
435:      this._DS_C_CONTACT = DS_C_CONTACT;
436:  }
437:
438:  /**
439:   * Sets the value of field 'DS_C_EMAIL'. The field 'DS_C_EMAIL'
440:   * has the following description: Email address of Data Source
441:   * clinical representative
442:   *
443:   * @param DS_C_EMAIL the value of field 'DS_C_EMAIL'.
444:   */
445:  public void setDS_C_EMAIL(
446:      final java.lang.String DS_C_EMAIL) {
447:      this._DS_C_EMAIL = DS_C_EMAIL;
448:  }
449:
```

```
450:  /**
451:   * Sets the value of field 'DS_ID'. The field 'DS_ID' has the
452:   * following description: Unique centre identification number
453:   * (Defined as a BIRO Clinical Site)
454:   *
455:   * @param DS_ID the value of field 'DS_ID'.
456:   */
457:  public void setDS_ID(
458:      final DataSource DS_ID) {
459:      this._DS_ID = DS_ID;
460:  }
461:
462:  /**
463:   * Sets the value of field 'DS_POST_CODE'. The field
464:   * 'DS_POST_CODE' has the following description: Post Code of
465:   * Data Source
466:   *
467:   * @param DS_POST_CODE the value of field 'DS_POST_CODE'.
468:   */
469:  public void setDS_POST_CODE(
470:      final java.lang.String DS_POST_CODE) {
471:      this._DS_POST_CODE = DS_POST_CODE;
472:  }
473:
474:  /**
475:   * Sets the value of field 'DS_T_CONTACT'. The field
476:   * 'DS_T_CONTACT' has the following description: Technical
477:   * representative from Data Source
478:   *
479:   * @param DS_T_CONTACT the value of field 'DS_T_CONTACT'.
480:   */
481:  public void setDS_T_CONTACT(
482:      final java.lang.String DS_T_CONTACT) {
483:      this._DS_T_CONTACT = DS_T_CONTACT;
484:  }
485:
486:  /**
487:   * Sets the value of field 'DS_T_EMAIL'. The field 'DS_T_EMAIL'
488:   * has the following description: Email address of Data Source
489:   * technical representative
490:   *
491:   * @param DS_T_EMAIL the value of field 'DS_T_EMAIL'.
492:   */
493:  public void setDS_T_EMAIL(
494:      final java.lang.String DS_T_EMAIL) {
```

```
495:         this._DS_T_EMAIL = DS_T_EMAIL;
496:     }
497:
498:     /**
499:     * Sets the value of field 'DS_WEBSITE'. The field 'DS_WEBSITE'
500:     * has the following description: Internet address for Data
501:     * Source
502:     *
503:     * @param DS_WEBSITE the value of field 'DS_WEBSITE'.
504:     */
505:     public void setDS_WEBSITE(
506:         final java.lang.String DS_WEBSITE) {
507:         this._DS_WEBSITE = DS_WEBSITE;
508:     }
509:
510:     /**
511:     * Sets the value of field 'hibernateDateHeaderInformationChecked'.
512:     *
513:     * @param hibernateDateHeaderInformationChecked the value of field
514:     * 'hibernateDateHeaderInformationChecked'.
515:     */
516:     public void setHibernateDateHeaderInformationChecked(
517:         final java.util.Date hibernateDateHeaderInformationChecked) {
518:         this._hibernateDateHeaderInformationChecked = hibernateDateHeaderInformationChecked;
519:     }
520:
521:     /**
522:     * Sets the value of field 'dateHeaderInformationChecked'.
523:     *
524:     * @param dateHeaderInformationChecked the value of field
525:     * 'dateHeaderInformationChecked'.
526:     */
527:     public void setDateHeaderInformationChecked(
528:         final org.exolab.castor.types.Date dateHeaderInformationChecked) {
529:         this._dateHeaderInformationChecked = dateHeaderInformationChecked;
530:     }
531:
532:     /**
533:     * Sets the value of field 'headerComments'.
534:     *
535:     * @param headerComments the value of field 'headerComments'.
536:     */
537:     public void setHeaderComments(
538:         final java.lang.String headerComments) {
539:         this._headerComments = headerComments;
```

```
540:     }
541:
542:     /**
543:      * Method unmarshal.
544:      *
545:      * @param reader
546:      * @throws org.exolab.castor.xml.MarshalException if object is
547:      * null or if any SAXException is thrown during marshaling
548:      * @throws org.exolab.castor.xml.ValidationException if this
549:      * object is an invalid instance according to the schema
550:      * @return the unmarshaled DataSourceExport.SiteHeader
551:      */
552:     public static export.SiteHeader unmarshal(
553:         final java.io.Reader reader)
554:         throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
555:         return (export.SiteHeader) Unmarshaller.unmarshal(export.SiteHeader.class, reader);
556:     }
557:
558:     /**
559:      *
560:      *
561:      * @throws org.exolab.castor.xml.ValidationException if this
562:      * object is an invalid instance according to the schema
563:      */
564:     public void validate()
565:         throws org.exolab.castor.xml.ValidationException {
566:         org.exolab.castor.xml.Validator validator = new org.exolab.castor.xml.Validator();
567:         validator.validate(this);
568:     }
569: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      SiteProfile.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10: * the Free Software Foundation; either version 2, or (at your option)
11: * any later version.
12: *
13: * This file is distributed in the hope that it will be useful,
14: * but WITHOUT ANY WARRANTY; without even the implied warranty of
15: * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16: * GNU General Public License for more details.
17: *
18: * You should have received a copy of the GNU General Public License
19: * along with this file; see the file COPYING. If not, write to
20: * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21: *
22: * In short: you may use this file any way you like, as long as you
23: * don't charge money for it, remove this notice, or hold anyone liable
24: * for its results.
25: *
26:
27: * GPL Copyright, The BIRO Project
28: *
29: **/
30: /*
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36:
37: package export;
38:
39:  //-----/
40:  //- Imported classes and packages -/
41:  //-----/
42:
43: import org.exolab.castor.xml.Marshaller;
44: import org.exolab.castor.xml.Unmarshaller;
45:
```

```
46: /**
47:  * Class SiteProfile.
48:  *
49:  * @version $Revision$ $Date$
50:  */
51: public class SiteProfile implements java.io.Serializable {
52:
53:
54:     //-----/
55:     //- Class/Member Variables -/
56:     //-----/
57:     /**
58:     Aggiunto da Valentina
59:     */
60:     private int _ID =-1;
61:     /**
62:     * Field _dateProfileInformationChecked.
63:     */
64:     private org.exolab.castor.types.Date _dateProfileInformationChecked;
65:
66:     /**
67:     * Field _hibernateDateProfileInformationChecked.
68:     */
69:     private java.util.Date _hibernateDateProfileInformationChecked;
70:
71:     /**
72:     * Field _DS_TYPE.
73:     */
74:     private export.types.SiteType _DS_TYPE;
75:
76:     /**
77:     * Current data source population (with or without diabetes)
78:     */
79:     private long _DS_DENOM;
80:
81:     /**
82:     * keeps track of state for field: _DS_DENOM
83:     */
84:     private boolean _has_DS_DENOM;
85:
86:     /**
87:     * The total population of patients with known diabetes in the
88:     * catchment area of the clinic
89:     */
90:     private long _DS_AREA;
```

```
91:
92:  /**
93:   * keeps track of state for field: _DS_AREA
94:   */
95:  private boolean _has_DS_AREA;
96:
97:  /**
98:   * Total hospital beds within data source geographical area -
99:   * not separated by category
100:  */
101:  private long _DS_BEDS;
102:
103:  /**
104:   * keeps track of state for field: _DS_BEDS
105:   */
106:  private boolean _has_DS_BEDS;
107:
108:  /**
109:   * Physicians within data source geographical area. National
110:   * statistics can provide information on this indicator
111:   */
112:  private long _DS_PHYSICIANS;
113:
114:  /**
115:   * keeps track of state for field: _DS_PHYSICIANS
116:   */
117:  private boolean _has_DS_PHYSICIANS;
118:
119:  /**
120:   * Diabetologists within data source geographical area. Data
121:   * should come from national Specialist Registers
122:   */
123:  private long _DS_DIABETOLOGISTS;
124:
125:  /**
126:   * keeps track of state for field: _DS_DIABETOLOGISTS
127:   */
128:  private boolean _has_DS_DIABETOLOGISTS;
129:
130:  /**
131:   * Number of doctors who regularly take care of diabetic
132:   * patients in diabetes clinics in primary or secondary care
133:   * within data source geographical area
134:   */
135:  private long _DS_DOCTORS;
```

```
136:
137: /**
138:  * keeps track of state for field: _DS_DOCTORS
139:  */
140: private boolean _has_DS_DOCTORS;
141:
142: /**
143:  * Specialist diabetes nurses within data source geographical
144:  * area
145:  */
146: private long _DS_DSN;
147:
148: /**
149:  * keeps track of state for field: _DS_DSN
150:  */
151: private boolean _has_DS_DSN;
152:
153: /**
154:  * The number of Physicians offering and recruiting for
155:  * structured Diabetes Disease Management Programmes
156:  */
157: private long _DS_DMP_PHYSICIANS;
158:
159: /**
160:  * keeps track of state for field: _DS_DMP_PHYSICIANS
161:  */
162: private boolean _has_DS_DMP_PHYSICIANS;
163:
164: /**
165:  * Field _profileComments.
166:  */
167: private java.lang.String _profileComments;
168:
169:
170: //-----/
171: //- Constructors -/
172: //-----/
173:
174: public SiteProfile() {
175:     super();
176: }
177:
178:
179: //-----/
180: //- Methods -/
```



```
181:  //-----/
182:
183:  /**
184:   * @param vID
185:   */
186:  public void setID(
187:      final int vID)
188:  {
189:      this._ID = vID;
190:  }
191:
192:  /**
193:   * @param vID
194:   */
195:  public int getID()
196:  {
197:      return this._ID;
198:  }
199:
200:
201:  /**
202:   */
203:  public void deleteDS_AREA(
204:  ) {
205:      this._has_DS_AREA= false;
206:  }
207:
208:  /**
209:   */
210:  public void deleteDS_BEDS(
211:  ) {
212:      this._has_DS_BEDS= false;
213:  }
214:
215:  /**
216:   */
217:  public void deleteDS_DENOM(
218:  ) {
219:      this._has_DS_DENOM= false;
220:  }
221:
222:  /**
223:   */
224:  public void deleteDS_DIABETOLOGISTS(
225:  ) {
```

```
226:         this._has_DS_DIABETOLOGISTS= false;
227:     }
228:
229:     /**
230:      */
231:     public void deleteDS_DOCTORS(
232:     ) {
233:         this._has_DS_DOCTORS= false;
234:     }
235:
236:     /**
237:      */
238:     public void deleteDS_DSN(
239:     ) {
240:         this._has_DS_DSN= false;
241:     }
242:
243:     /**
244:      */
245:     public void deleteDS_PHYSICIANS(
246:     ) {
247:         this._has_DS_PHYSICIANS= false;
248:     }
249:
250:     /**
251:      */
252:     public void deleteDS_DMP_PHYSICIANS(
253:     ) {
254:         this._has_DS_DMP_PHYSICIANS= false;
255:     }
256:
257:     /**
258:      * Returns the value of field 'DS_AREA'. The field 'DS_AREA'
259:      * has the following description: The total population of
260:      * patients with known diabetes in the catchment area of the
261:      * clinic
262:      *
263:      * @return the value of field 'DS_AREA'.
264:      */
265:     public long getDS_AREA(
266:     ) {
267:         return this._DS_AREA;
268:     }
269:
270:     /**
```

```
271:      * Returns the value of field 'DS_BEDS'. The field 'DS_BEDS'
272:      * has the following description: Total hospital beds within
273:      * data source geographical area - not separated by category
274:      *
275:      * @return the value of field 'DS_BEDS'.
276:      */
277: public long getDS_BEDS(
278: ) {
279:     return this._DS_BEDS;
280: }
281:
282: /**
283:  * Returns the value of field 'DS_DENOM'. The field 'DS_DENOM'
284:  * has the following description: Current data source
285:  * population (with or without diabetes)
286:  *
287:  * @return the value of field 'DS_DENOM'.
288:  */
289: public long getDS_DENOM(
290: ) {
291:     return this._DS_DENOM;
292: }
293:
294: /**
295:  * Returns the value of field 'DS_DIABETOLOGISTS'. The field
296:  * 'DS_DIABETOLOGISTS' has the following description:
297:  * Diabetologists within data source geographical area. Data
298:  * should come from national Specialist Registers
299:  *
300:  * @return the value of field 'DS_DIABETOLOGISTS'.
301:  */
302: public long getDS_DIABETOLOGISTS(
303: ) {
304:     return this._DS_DIABETOLOGISTS;
305: }
306:
307: /**
308:  * Returns the value of field 'DS_DOCTORS'. The field
309:  * 'DS_DOCTORS' has the following description: Number of
310:  * doctors who regularly take care of diabetic patients in
311:  * diabetes clinics in primary or secondary care within data
312:  * source geographical area
313:  *
314:  * @return the value of field 'DS_DOCTORS'.
315:  */
```

```
316: public long getDS_DOCTORS(
317: ) {
318:     return this._DS_DOCTORS;
319: }
320:
321: /**
322:  * Returns the value of field 'DS_DSN'. The field 'DS_DSN' has
323:  * the following description: Specialist diabetes nurses within
324:  * data source geographical area
325:  *
326:  * @return the value of field 'DS_DSN'.
327:  */
328: public long getDS_DSN(
329: ) {
330:     return this._DS_DSN;
331: }
332:
333: /**
334:  * Returns the value of field 'DS_PHYSICIANS'. The field
335:  * 'DS_PHYSICIANS' has the following description: Physicians
336:  * within data source geographical area. National statistics
337:  * can provide information on this indicator
338:  *
339:  * @return the value of field 'DS_PHYSICIANS'.
340:  */
341: public long getDS_PHYSICIANS(
342: ) {
343:     return this._DS_PHYSICIANS;
344: }
345:
346: /**
347:  * Returns the value of field 'DS_DMP_PHYSICIANS'.
348:  *
349:  * @return the value of field 'DS_DMP_PHYSICIANS'.
350:  */
351: public long getDS_DMP_PHYSICIANS(
352: ) {
353:     return this._DS_DMP_PHYSICIANS;
354: }
355:
356: /**
357:  * Returns the value of field 'DS_TYPE'.
358:  *
359:  * @return the value of field 'DS_TYPE'.
360:  */
```

```
361: public export.types.SiteType getDS_TYPE(  
362: ) {  
363:     return this._DS_TYPE;  
364: }  
365:  
366: /**  
367:  * Returns the value of field 'dateProfileInformationChecked'.  
368:  *  
369:  * @return the value of field 'DateProfileInformationChecked'.  
370:  */  
371: public org.exolab.castor.types.Date getDateProfileInformationChecked(  
372: ) {  
373:     return this._dateProfileInformationChecked;  
374: }  
375:  
376: /**  
377:  * Returns the value of field 'hibernateDateProfileInformationChecked'.  
378:  *  
379:  * @return the value of field 'hibernateDateProfileInformationChecked'.  
380:  */  
381: public java.util.Date getHibernateDateProfileInformationChecked(  
382: ) {  
383:     this._hibernateDateProfileInformationChecked = this._dateProfileInformationChecked.toDate();  
384:     return this._hibernateDateProfileInformationChecked;  
385: }  
386:  
387: /**  
388:  * Returns the value of field 'profileComments'.  
389:  *  
390:  * @return the value of field 'ProfileComments'.  
391:  */  
392: public java.lang.String getProfileComments(  
393: ) {  
394:     return this._profileComments;  
395: }  
396:  
397: /**  
398:  * Method hasDS_AREA.  
399:  *  
400:  * @return true if at least one DS_AREA has been added  
401:  */  
402: public boolean hasDS_AREA(  
403: ) {  
404:     return this._has_DS_AREA;  
405: }
```

```
406:
407: /**
408:  * Method hasDS_BEDS.
409:  *
410:  * @return true if at least one DS_BEDS has been added
411:  */
412: public boolean hasDS_BEDS(
413: ) {
414:     return this._has_DS_BEDS;
415: }
416:
417: /**
418:  * Method hasDS_DENOM.
419:  *
420:  * @return true if at least one DS_DENOM has been added
421:  */
422: public boolean hasDS_DENOM(
423: ) {
424:     return this._has_DS_DENOM;
425: }
426:
427: /**
428:  * Method hasDS_DIABETOLOGISTS.
429:  *
430:  * @return true if at least one DS_DIABETOLOGISTS has been added
431:  */
432: public boolean hasDS_DIABETOLOGISTS(
433: ) {
434:     return this._has_DS_DIABETOLOGISTS;
435: }
436:
437: /**
438:  * Method hasDS_DOCTORS.
439:  *
440:  * @return true if at least one DS_DOCTORS has been added
441:  */
442: public boolean hasDS_DOCTORS(
443: ) {
444:     return this._has_DS_DOCTORS;
445: }
446:
447: /**
448:  * Method hasDS_DSN.
449:  *
450:  * @return true if at least one DS_DSN has been added
```

```
451:      */
452:      public boolean hasDS_DSN(
453:      ) {
454:          return this._has_DS_DSN;
455:      }
456:
457:      /**
458:       * Method hasDS_PHYSICIANS.
459:       *
460:       * @return true if at least one DS_PHYSICIANS has been added
461:       */
462:      public boolean hasDS_PHYSICIANS(
463:      ) {
464:          return this._has_DS_PHYSICIANS;
465:      }
466:
467:      /**
468:       * Method hasDS_DMP_PHYSICIANS.
469:       *
470:       * @return true if at least one DS_DMP_PHYSICIANS has been added
471:       */
472:      public boolean hasDS_DMP_PHYSICIANS(
473:      ) {
474:          return this._has_DS_DMP_PHYSICIANS;
475:      }
476:
477:      /**
478:       * Method isValid.
479:       *
480:       * @return true if this object is valid according to the schema
481:       */
482:      public boolean isValid(
483:      ) {
484:          try {
485:              validate();
486:          } catch (org.exolab.castor.xml.ValidationException vex) {
487:              return false;
488:          }
489:          return true;
490:      }
491:
492:      /**
493:       *
494:       *
495:       * @param out
```

```
496:      * @throws org.exolab.castor.xml.MarshalException if object is
497:      * null or if any SAXException is thrown during marshaling
498:      * @throws org.exolab.castor.xml.ValidationException if this
499:      * object is an invalid instance according to the schema
500:      */
501:  public void marshal(
502:      final java.io.Writer out)
503:  throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
504:      Marshaller.marshal(this, out);
505:  }
506:
507:  /**
508:   *
509:   *
510:   * @param handler
511:   * @throws java.io.IOException if an IOException occurs during
512:   * marshaling
513:   * @throws org.exolab.castor.xml.ValidationException if this
514:   * object is an invalid instance according to the schema
515:   * @throws org.exolab.castor.xml.MarshalException if object is
516:   * null or if any SAXException is thrown during marshaling
517:   */
518:  public void marshal(
519:      final org.xml.sax.ContentHandler handler)
520:  throws java.io.IOException, org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
521:      Marshaller.marshal(this, handler);
522:  }
523:
524:  /**
525:   * Sets the value of field 'DS_AREA'. The field 'DS_AREA' has
526:   * the following description: The total population of patients
527:   * with known diabetes in the catchment area of the clinic
528:   *
529:   * @param DS_AREA the value of field 'DS_AREA'.
530:   */
531:  public void setDS_AREA(
532:      final long DS_AREA) {
533:      this._DS_AREA = DS_AREA;
534:      this._has_DS_AREA = true;
535:  }
536:
537:  /**
538:   * Sets the value of field 'DS_BEDS'. The field 'DS_BEDS' has
539:   * the following description: Total hospital beds within data
540:   * source geographical area - not separated by category
```



```
541:      *
542:      * @param DS_BEDS the value of field 'DS_BEDS'.
543:      */
544:      public void setDS_BEDS(
545:          final long DS_BEDS) {
546:          this._DS_BEDS = DS_BEDS;
547:          this._has_DS_BEDS = true;
548:      }
549:
550:      /**
551:       * Sets the value of field 'DS_DENOM'. The field 'DS_DENOM' has
552:       * the following description: Current data source population
553:       * (with or without diabetes)
554:       *
555:       * @param DS_DENOM the value of field 'DS_DENOM'.
556:       */
557:      public void setDS_DENOM(
558:          final long DS_DENOM) {
559:          this._DS_DENOM = DS_DENOM;
560:          this._has_DS_DENOM = true;
561:      }
562:
563:      /**
564:       * Sets the value of field 'DS_DIABETOLOGISTS'. The field
565:       * 'DS_DIABETOLOGISTS' has the following description:
566:       * Diabetologists within data source geographical area. Data
567:       * should come from national Specialist Registers
568:       *
569:       * @param DS_DIABETOLOGISTS the value of field
570:       * 'DS_DIABETOLOGISTS'.
571:       */
572:      public void setDS_DIABETOLOGISTS(
573:          final long DS_DIABETOLOGISTS) {
574:          this._DS_DIABETOLOGISTS = DS_DIABETOLOGISTS;
575:          this._has_DS_DIABETOLOGISTS = true;
576:      }
577:
578:      /**
579:       * Sets the value of field 'DS_DOCTORS'. The field 'DS_DOCTORS'
580:       * has the following description: Number of doctors who
581:       * regularly take care of diabetic patients in diabetes clinics
582:       * in primary or secondary care within data source geographical
583:       * area
584:       *
585:       * @param DS_DOCTORS the value of field 'DS_DOCTORS'.
```

```
586:    */
587:    public void setDS_DOCTORS(
588:        final long DS_DOCTORS) {
589:        this._DS_DOCTORS = DS_DOCTORS;
590:        this._has_DS_DOCTORS = true;
591:    }
592:
593:    /**
594:     * Sets the value of field 'DS_DSN'. The field 'DS_DSN' has the
595:     * following description: Specialist diabetes nurses within
596:     * data source geographical area
597:     *
598:     * @param DS_DSN the value of field 'DS_DSN'.
599:     */
600:    public void setDS_DSN(
601:        final long DS_DSN) {
602:        this._DS_DSN = DS_DSN;
603:        this._has_DS_DSN = true;
604:    }
605:
606:    /**
607:     * Sets the value of field 'DS_PHYSICIANS'. The field
608:     * 'DS_PHYSICIANS' has the following description: Physicians
609:     * within data source geographical area. National statistics
610:     * can provide information on this indicator
611:     *
612:     * @param DS_PHYSICIANS the value of field 'DS_PHYSICIANS'.
613:     */
614:    public void setDS_PHYSICIANS(
615:        final long DS_PHYSICIANS) {
616:        this._DS_PHYSICIANS = DS_PHYSICIANS;
617:        this._has_DS_PHYSICIANS = true;
618:    }
619:
620:    /**
621:     * Sets the value of field 'DS_DMP_PHYSICIANS'.
622:     * @param DS_DMP_PHYSICIANS the value of field 'DS_DMP_PHYSICIANS'.
623:     */
624:    public void setDS_DMP_PHYSICIANS(
625:        final long DS_DMP_PHYSICIANS) {
626:        this._DS_DMP_PHYSICIANS = DS_DMP_PHYSICIANS;
627:        this._has_DS_DMP_PHYSICIANS = true;
628:    }
629:
630:    /**
```

```
631:      * Sets the value of field 'DS_TYPE'.
632:      *
633:      * @param DS_TYPE the value of field 'DS_TYPE'.
634:      */
635:  public void setDS_TYPE(
636:      final export.types.SiteType DS_TYPE) {
637:      this._DS_TYPE = DS_TYPE;
638:  }
639:
640:  /**
641:   * Sets the value of field 'dateProfileInformationChecked'.
642:   *
643:   * @param dateProfileInformationChecked the value of field
644:   *   'dateProfileInformationChecked'.
645:   */
646:  public void setDateProfileInformationChecked(
647:      final org.exolab.castor.types.Date dateProfileInformationChecked) {
648:      this._dateProfileInformationChecked = dateProfileInformationChecked;
649:  }
650:
651:  /**
652:   * Sets the value of field 'hibernateDateProfileInformationChecked'.
653:   *
654:   * @param hibernateDateProfileInformationChecked the value of field
655:   *   'hibernateDateProfileInformationChecked'.
656:   */
657:  public void setHibernateDateProfileInformationChecked(
658:      final java.util.Date hibernateDateProfileInformationChecked) {
659:      this._hibernateDateProfileInformationChecked = hibernateDateProfileInformationChecked;
660:  }
661:
662:  /**
663:   * Sets the value of field 'profileComments'.
664:   *
665:   * @param profileComments the value of field 'profileComments'.
666:   */
667:  public void setProfileComments(
668:      final java.lang.String profileComments) {
669:      this._profileComments = profileComments;
670:  }
671:
672:  /**
673:   * Method unmarshal.
674:   *
675:   * @param reader
```

```
676:      * @throws org.exolab.castor.xml.MarshalException if object is
677:      * null or if any SAXException is thrown during marshaling
678:      * @throws org.exolab.castor.xml.ValidationException if this
679:      * object is an invalid instance according to the schema
680:      * @return the unmarshaled DataSourceExport.SiteProfile
681:      */
682:      public static export.SiteProfile unmarshal(
683:          final java.io.Reader reader)
684:          throws org.exolab.castor.xml.MarshalException, org.exolab.castor.xml.ValidationException {
685:          return (export.SiteProfile) Unmarshaller.unmarshal(export.SiteProfile.class, reader);
686:      }
687:
688:      /**
689:      *
690:      *
691:      * @throws org.exolab.castor.xml.ValidationException if this
692:      * object is an invalid instance according to the schema
693:      */
694:      public void validate(
695:      )
696:      throws org.exolab.castor.xml.ValidationException {
697:          org.exolab.castor.xml.Validator validator = new org.exolab.castor.xml.Validator();
698:          validator.validate(this);
699:      }
700:
701: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      ActivityDataDescriptor.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: /*
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36:
37: package export.descriptors;
38:
39:  //-----/
40:  //- Imported classes and packages -/
41:  //-----/
42:
43: import export.ActivityData;
44: import export.types.ActivityEndReason;
45: import export.types.ActivityStartReason;
```

```
46:
47: /**
48:  * Class ActivityDataDescriptor.
49:  *
50:  * @version $Revision$ $Date$
51:  */
52: public class ActivityDataDescriptor extends org.exolab.castor.xml.util.XMLClassDescriptorImpl {
53:
54:
55:     //-----/
56:     //- Class/Member Variables -/
57:     //-----/
58:
59:     /**
60:      * Field _elementDefinition.
61:      */
62:     private boolean _elementDefinition;
63:
64:     /**
65:      * Field _nsPrefix.
66:      */
67:     private java.lang.String _nsPrefix;
68:
69:     /**
70:      * Field _nsURI.
71:      */
72:     private java.lang.String _nsURI;
73:
74:     /**
75:      * Field _xmlName.
76:      */
77:     private java.lang.String _xmlName;
78:
79:     /**
80:      * Field _identity.
81:      */
82:     private org.exolab.castor.xml.XMLFieldDescriptor _identity;
83:
84:
85:     //-----/
86:     //- Constructors -/
87:     //-----/
88:
89:     public ActivityDataDescriptor() {
90:         super();
```

```
91:         _xmlName = "ActivityData";
92:         _elementDefinition = true;
93:
94:         //-- set grouping compositor
95:         setCompositorAsSequence();
96:         org.exolab.castor.xml.util.XMLFieldDescriptorImpl desc          = null;
97:         org.exolab.castor.mapping.FieldHandler handler              = null;
98:         org.exolab.castor.xml.FieldValidator fieldValidator = null;
99:         //-- initialize attribute descriptors
100:
101:         //-- initialize element descriptors
102:
103:         //-- _AD_START_DATE
104:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(org.exolab.castor.types.Date.class,
"AD_START_DATE", "AD_START_DATE", org.exolab.castor.xml.NodeType.Element);
105:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
106:             public java.lang.Object getValue( java.lang.Object object )
107:                 throws IllegalStateException
108:             {
109:                 ActivityData target = (ActivityData) object;
110:                 return target.getAD_START_DATE();
111:             }
112:             public void setValue( java.lang.Object object, java.lang.Object value)
113:                 throws IllegalStateException, IllegalArgumentException
114:             {
115:                 try {
116:                     ActivityData target = (ActivityData) object;
117:                     target.setAD_START_DATE( (org.exolab.castor.types.Date) value);
118:                 } catch (java.lang.Exception ex) {
119:                     throw new IllegalStateException(ex.toString());
120:                 }
121:             }
122:             public java.lang.Object newInstance(java.lang.Object parent) {
123:                 return new org.exolab.castor.types.Date();
124:             }
125:         };
126:         desc.setHandler(handler);
127:         //desc.setRequired(true);
128:         desc.setMultivalued(false);
129:         addFieldDescriptor(desc);
130:
131:         //-- validation code for: _AD_START_DATE
132:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
133:         //fieldValidator.setMinOccurs(1);
134:         fieldValidator.setMinOccurs(0);
```

BIRODatabaseManager/src/export/descriptors/ActivityDataDescriptor.java

```
135:         { //-- local scope
136:             org.exolab.castor.xml.validators.DateTimeValidator typeValidator;
137:             typeValidator = new org.exolab.castor.xml.validators.DateTimeValidator();
138:             fieldValidator.setValidator(typeValidator);
139:         }
140:         desc.setValidator(fieldValidator);
141:         //-- _AD_START_REASON
142:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(ActivityStartReason.class, "_AD_START_REASON"
, "AD_START_REASON", org.exolab.castor.xml.NodeType.Element);
143:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
144:             public java.lang.Object getValue( java.lang.Object object )
145:                 throws IllegalStateException
146:             {
147:                 ActivityData target = (ActivityData) object;
148:                 return target.getAD_START_REASON();
149:             }
150:             public void setValue( java.lang.Object object, java.lang.Object value)
151:                 throws IllegalStateException, IllegalArgumentException
152:             {
153:                 try {
154:                     ActivityData target = (ActivityData) object;
155:                     target.setAD_START_REASON( (export.types.ActivityStartReason) value);
156:                 } catch (java.lang.Exception ex) {
157:                     throw new IllegalStateException(ex.toString());
158:                 }
159:             }
160:             public java.lang.Object newInstance(java.lang.Object parent) {
161:                 return null;
162:             }
163:         };
164:         handler = new org.exolab.castor.xml.handlers.EnumFieldHandler(ActivityStartReason.class, handler);
165:         desc.setImmutable(true);
166:         desc.setHandler(handler);
167:         //desc.setRequired(true);
168:         desc.setMultivalued(false);
169:         addFieldDescriptor(desc);
170:
171:         //-- validation code for: _AD_START_REASON
172:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
173:         //fieldValidator.setMinOccurs(1);
174:         fieldValidator.setMinOccurs(0);
175:         { //-- local scope
176:         }
177:         desc.setValidator(fieldValidator);
178:         //-- _AD_END_DATE
```


BIRODatabaseManager/src/export/descriptors/ActivityDataDescriptor.java

```
179:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(org.exolab.castor.types.Date.class,  
"AD_END_DATE", "AD_END_DATE", org.exolab.castor.xml.NodeType.Element);  
180:         handler = new org.exolab.castor.xml.XMLFieldHandler() {  
181:             public java.lang.Object getValue( java.lang.Object object )  
182:                 throws IllegalStateException  
183:             {  
184:                 ActivityData target = (ActivityData) object;  
185:                 return target.getAD_END_DATE();  
186:             }  
187:             public void setValue( java.lang.Object object, java.lang.Object value)  
188:                 throws IllegalStateException, IllegalArgumentException  
189:             {  
190:                 try {  
191:                     ActivityData target = (ActivityData) object;  
192:                     target.setAD_END_DATE( (org.exolab.castor.types.Date) value);  
193:                 } catch (java.lang.Exception ex) {  
194:                     throw new IllegalStateException(ex.toString());  
195:                 }  
196:             }  
197:             public java.lang.Object newInstance(java.lang.Object parent) {  
198:                 return new org.exolab.castor.types.Date();  
199:             }  
200:         };  
201:         desc.setHandler(handler);  
202:         desc.setMultivalued(false);  
203:         addFieldDescriptor(desc);  
204:  
205:         //-- validation code for: _AD_END_DATE  
206:         fieldValidator = new org.exolab.castor.xml.FieldValidator();  
207:         { //-- local scope  
208:             org.exolab.castor.xml.validators.DateTimeValidator typeValidator;  
209:             typeValidator = new org.exolab.castor.xml.validators.DateTimeValidator();  
210:             fieldValidator.setValidator(typeValidator);  
211:         }  
212:         desc.setValidator(fieldValidator);  
213:         //-- _AD_END_REASON  
214:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(ActivityEndReason.class, "_AD_END_REASON",  
"AD_END_REASON", org.exolab.castor.xml.NodeType.Element);  
215:         handler = new org.exolab.castor.xml.XMLFieldHandler() {  
216:             public java.lang.Object getValue( java.lang.Object object )  
217:                 throws IllegalStateException  
218:             {  
219:                 ActivityData target = (ActivityData) object;  
220:                 return target.getAD_END_REASON();  
221:             }  
222:         }  
223:     }  
224: }
```

```
222:         public void setValue( java.lang.Object object, java.lang.Object value)
223:             throws IllegalStateException, IllegalArgumentException
224:         {
225:             try {
226:                 ActivityData target = (ActivityData) object;
227:                 target.setAD_END_REASON( (ActivityEndReason) value);
228:             } catch (java.lang.Exception ex) {
229:                 throw new IllegalStateException(ex.toString());
230:             }
231:         }
232:         public java.lang.Object newInstance(java.lang.Object parent) {
233:             return null;
234:         }
235:     };
236:     handler = new org.exolab.castor.xml.handlers.EnumFieldHandler(ActivityEndReason.class, handler);
237:     desc.setImmutable(true);
238:     desc.setHandler(handler);
239:     desc.setMultivalued(false);
240:     addFieldDescriptor(desc);
241:
242:     //-- validation code for: _AD_END_REASON
243:     fieldValidator = new org.exolab.castor.xml.FieldValidator();
244:     { //-- local scope
245:     }
246:     desc.setValidator(fieldValidator);
247: }
248:
249:
250:     //-----/
251:     //- Methods -/
252:     //-----/
253:
254:     /**
255:      * Method getAccessMode.
256:      *
257:      * @return the access mode specified for this class.
258:      */
259:     public org.exolab.castor.mapping.AccessMode getAccessMode(
260:     ) {
261:         return null;
262:     }
263:
264:     /**
265:      * Method getIdentity.
266:      *
```

```
267:      * @return the identity field, null if this class has no
268:      * identity.
269:      */
270: public org.exolab.castor.mapping.FieldDescriptor getIdentity(
271: ) {
272:     return _identity;
273: }
274:
275: /**
276:  * Method getJavaClass.
277:  *
278:  * @return the Java class represented by this descriptor.
279:  */
280: public java.lang.Class getJavaClass(
281: ) {
282:     return ActivityData.class;
283: }
284:
285: /**
286:  * Method getNameSpacePrefix.
287:  *
288:  * @return the namespace prefix to use when marshaling as XML.
289:  */
290: public java.lang.String getNameSpacePrefix(
291: ) {
292:     return _nsPrefix;
293: }
294:
295: /**
296:  * Method getNameSpaceURI.
297:  *
298:  * @return the namespace URI used when marshaling and
299:  * unmarshaling as XML.
300:  */
301: public java.lang.String getNameSpaceURI(
302: ) {
303:     return _nsURI;
304: }
305:
306: /**
307:  * Method getValidator.
308:  *
309:  * @return a specific validator for the class described by this
310:  * ClassDescriptor.
311:  */
```

```
312: public org.exolab.castor.xml.TypeValidator getValidator(  
313: ) {  
314:     return this;  
315: }  
316:  
317: /**  
318:  * Method getXMLName.  
319:  *  
320:  * @return the XML Name for the Class being described.  
321:  */  
322: public java.lang.String getXMLName(  
323: ) {  
324:     return _xmlName;  
325: }  
326:  
327: /**  
328:  * Method isElementDefinition.  
329:  *  
330:  * @return true if XML schema definition of this Class is that  
331:  * of a global  
332:  * element or element with anonymous type definition.  
333:  */  
334: public boolean isElementDefinition(  
335: ) {  
336:     return _elementDefinition;  
337: }  
338:  
339: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      DataDescriptor.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: /*
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36:
37: package export.descriptors;
38:
39:  //-----/
40:  //- Imported classes and packages -/
41:  //-----/
42:
43: import export.Data;
44:
45: /**
```

```
46:  * Class DataDescriptor.
47:  *
48:  * @version $Revision$ $Date$
49:  */
50: public class DataDescriptor extends org.exolab.castor.xml.util.XMLClassDescriptorImpl {
51:
52:
53:     //-----/
54:     //- Class/Member Variables -/
55:     //-----/
56:
57:     /**
58:      * Field _elementDefinition.
59:      */
60:     private boolean _elementDefinition;
61:
62:     /**
63:      * Field _nsPrefix.
64:      */
65:     private java.lang.String _nsPrefix;
66:
67:     /**
68:      * Field _nsURI.
69:      */
70:     private java.lang.String _nsURI;
71:
72:     /**
73:      * Field _xmlName.
74:      */
75:     private java.lang.String _xmlName;
76:
77:     /**
78:      * Field _identity.
79:      */
80:     private org.exolab.castor.xml.XMLFieldDescriptor _identity;
81:
82:
83:     //-----/
84:     //- Constructors -/
85:     //-----/
86:
87:     public DataDescriptor() {
88:         super();
89:         _xmlName = "Data";
90:         _elementDefinition = true;
```

```
91:
92:     //-- set grouping compositor
93:     setCompositorAsSequence();
94:     org.exolab.castor.xml.util.XMLFieldDescriptorImpl desc          = null;
95:     org.exolab.castor.mapping.FieldHandler handler                = null;
96:     org.exolab.castor.xml.FieldValidator fieldValidator          = null;
97:     //-- initialize attribute descriptors
98:
99:     //-- initialize element descriptors
100:
101:     //-- _episodeFieldName
102:     desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(export.types.BIRODataSet.class,
"episodeFieldName", "EpisodeFieldName", org.exolab.castor.xml.NodeType.Element);
103:     handler = new org.exolab.castor.xml.XMLFieldHandler() {
104:         public java.lang.Object getValue( java.lang.Object object )
105:             throws IllegalStateException
106:         {
107:             Data target = (Data) object;
108:             return target.getEpisodeFieldName();
109:         }
110:         public void setValue( java.lang.Object object, java.lang.Object value)
111:             throws IllegalStateException, IllegalArgumentException
112:         {
113:             try {
114:                 Data target = (Data) object;
115:                 target.setEpisodeFieldName( (export.types.BIRODataSet) value);
116:             } catch (java.lang.Exception ex) {
117:                 throw new IllegalStateException(ex.toString());
118:             }
119:         }
120:         public java.lang.Object newInstance(java.lang.Object parent) {
121:             return null;
122:         }
123:     };
124:     handler = new org.exolab.castor.xml.handlers.EnumFieldHandler(export.types.BIRODataSet.class, handler);
125:     desc.setImmutable(true);
126:     desc.setHandler(handler);
127:     desc.setRequired(true);
128:     desc.setMultivalued(false);
129:     addFieldDescriptor(desc);
130:
131:     //-- validation code for: _episodeFieldName
132:     fieldValidator = new org.exolab.castor.xml.FieldValidator();
133:     fieldValidator.setMinOccurs(1);
134:     { //-- local scope
```

```
135:         }
136:         desc.setValidator(fieldValidator);
137:         //-- _episodeFieldValue
138:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.String.class, "_episodeFieldValue",
"EpisodeFieldValue", org.exolab.castor.xml.NodeType.Element);
139:         desc.setImmutable(true);
140:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
141:             public java.lang.Object getValue( java.lang.Object object )
142:                 throws IllegalStateException
143:             {
144:                 Data target = (Data) object;
145:                 return target.getEpisodeFieldValue();
146:             }
147:             public void setValue( java.lang.Object object, java.lang.Object value)
148:                 throws IllegalStateException, IllegalArgumentException
149:             {
150:                 try {
151:                     Data target = (Data) object;
152:                     target.setEpisodeFieldValue( (java.lang.String) value);
153:                 } catch (java.lang.Exception ex) {
154:                     throw new IllegalStateException(ex.toString());
155:                 }
156:             }
157:             public java.lang.Object newInstance(java.lang.Object parent) {
158:                 return null;
159:             }
160:         };
161:         desc.setHandler(handler);
162:         desc.setRequired(true);
163:         desc.setMultivalued(false);
164:         addFieldDescriptor(desc);
165:
166:         //-- validation code for: _episodeFieldValue
167:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
168:         fieldValidator.setMinOccurs(1);
169:         { //-- local scope
170:             org.exolab.castor.xml.validators.StringValidator typeValidator;
171:             typeValidator = new org.exolab.castor.xml.validators.StringValidator();
172:             fieldValidator.setValidator(typeValidator);
173:             typeValidator.setWhiteSpace("preserve");
174:         }
175:         desc.setValidator(fieldValidator);
176:     }
177:
178:
```



```
179:         //-----/
180:         //- Methods -/
181:         //-----/
182:
183:         /**
184:          * Method getAccessMode.
185:          *
186:          * @return the access mode specified for this class.
187:          */
188:         public org.exolab.castor.mapping.AccessMode getAccessMode(
189:         ) {
190:             return null;
191:         }
192:
193:         /**
194:          * Method getIdentity.
195:          *
196:          * @return the identity field, null if this class has no
197:          * identity.
198:          */
199:         public org.exolab.castor.mapping.FieldDescriptor getIdentity(
200:         ) {
201:             return _identity;
202:         }
203:
204:         /**
205:          * Method getJavaClass.
206:          *
207:          * @return the Java class represented by this descriptor.
208:          */
209:         public java.lang.Class getJavaClass(
210:         ) {
211:             return Data.class;
212:         }
213:
214:         /**
215:          * Method getNameSpacePrefix.
216:          *
217:          * @return the namespace prefix to use when marshaling as XML.
218:          */
219:         public java.lang.String getNameSpacePrefix(
220:         ) {
221:             return _nsPrefix;
222:         }
223:
```

```
224:  /**
225:   * Method getNamespaceURI.
226:   *
227:   * @return the namespace URI used when marshaling and
228:   * unmarshaling as XML.
229:   */
230: public java.lang.String getNamespaceURI(
231: ) {
232:     return _nsURI;
233: }
234:
235:  /**
236:   * Method getValidator.
237:   *
238:   * @return a specific validator for the class described by this
239:   * ClassDescriptor.
240:   */
241: public org.exolab.castor.xml.TypeValidator getValidator(
242: ) {
243:     return this;
244: }
245:
246:  /**
247:   * Method getXMLName.
248:   *
249:   * @return the XML Name for the Class being described.
250:   */
251: public java.lang.String getXMLName(
252: ) {
253:     return _xmlName;
254: }
255:
256:  /**
257:   * Method isElementDefinition.
258:   *
259:   * @return true if XML schema definition of this Class is that
260:   * of a global
261:   * element or element with anonymous type definition.
262:   */
263: public boolean isElementDefinition(
264: ) {
265:     return _elementDefinition;
266: }
267:
268: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      DataHeaderDscriptor.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * donâ200\231t charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  **/
30: /*
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36: package export.descriptors;
37:
38: //-----/
39: //- Imported classes and packages -/
40: //-----/
41: import export.DataHeader;
42: import export.SiteHeader;
43: import export.types.DataSource;
44:
45: /**
```

```
46:  * Class SiteHeaderDescriptor.
47:  *
48:  * @version $Revision$ $Date$
49:  */
50: public class DataHeaderDescriptor extends org.exolab.castor.xml.util.XMLClassDescriptorImpl {
51:
52:
53:     //-----/
54:     //- Class/Member Variables -/
55:     //-----/
56:     /**
57:      * Field _elementDefinition.
58:      */
59:     private boolean _elementDefinition;
60:     /**
61:      * Field _nsPrefix.
62:      */
63:     private java.lang.String _nsPrefix;
64:     /**
65:      * Field _nsURI.
66:      */
67:     private java.lang.String _nsURI;
68:     /**
69:      * Field _xmlName.
70:      */
71:     private java.lang.String _xmlName;
72:     /**
73:      * Field _identity.
74:      */
75:     private org.exolab.castor.xml.XMLFieldDescriptor _identity;
76:
77:
78:     //-----/
79:     //- Constructors -/
80:     //-----/
81:     public DataHeaderDescriptor() {
82:         super();
83:         _xmlName = "DataHeader";
84:         _elementDefinition = true;
85:
86:         //-- set grouping compositor
87:         setCompositorAsSequence();
88:         org.exolab.castor.xml.util.XMLFieldDescriptorImpl desc = null;
89:         org.exolab.castor.mapping.FieldHandler handler = null;
90:         org.exolab.castor.xml.FieldValidator fieldValidator = null;
```

```
91:         //-- initialize attribute descriptors
92:
93:         //-- initialize element descriptors
94:
95:         //-- _dateCreated
96:         //desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(org.exolab.castor.types.Date.class,
"_DateCreated", "DateCreated", org.exolab.castor.xml.NodeType.Element);
97:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.String.class, "_dateCreated",
"DateCreated", org.exolab.castor.xml.NodeType.Element);
98:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
99:
100:             public java.lang.Object getValue(java.lang.Object object)
101:                 throws IllegalStateException {
102:                 DataHeader target = (DataHeader) object;
103:                 return target.getDateCreated();
104:             }
105:
106:             public void setValue(java.lang.Object object, java.lang.Object value)
107:                 throws IllegalStateException, IllegalArgumentException {
108:                 try {
109:                     DataHeader target = (DataHeader) object;
110:                     target.setDateCreated((java.lang.String) value);
111:                 } catch (java.lang.Exception ex) {
112:                     throw new IllegalStateException(ex.toString());
113:                 }
114:             }
115:
116:             public java.lang.Object newInstance(java.lang.Object parent) {
117:                 //return new org.exolab.castor.types.Date();
118:                 return null;
119:             }
120:         };
121:         desc.setHandler(handler);
122:         desc.setRequired(true);
123:         desc.setMultivalued(false);
124:         addFieldDescriptor(desc);
125:
126:         //-- validation code for: _dateCreated
127:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
128:         fieldValidator.setMinOccurs(1);
129:         { //-- local scope
130:             /*
131:             org.exolab.castor.xml.validators.DateTimeValidator typeValidator;
132:             typeValidator = new org.exolab.castor.xml.validators.DateTimeValidator();
133:             fieldValidator.setValidator(typeValidator);
```

```
134:         */
135:         org.exolab.castor.xml.validators.StringValidator typeValidator;
136:         typeValidator = new org.exolab.castor.xml.validators.StringValidator();
137:         fieldValidator.setValidator(typeValidator);
138:         typeValidator.setWhiteSpace("preserve");
139:     }
140:     desc.setValidator(fieldValidator);
141:     //-- _DS_ID
142:     desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(DataSource.class, "_DS_ID", "DS_ID",
org.exolab.castor.xml.NodeType.Element);
143:     handler = new org.exolab.castor.xml.XMLFieldHandler() {
144:
145:         public java.lang.Object getValue(java.lang.Object object)
146:             throws IllegalStateException {
147:             DataHeader target = (DataHeader) object;
148:             return target.getDS_ID();
149:         }
150:
151:         public void setValue(java.lang.Object object, java.lang.Object value)
152:             throws IllegalStateException, IllegalArgumentException {
153:             try {
154:                 DataHeader target = (DataHeader) object;
155:                 target.setDS_ID((DataSource) value);
156:             } catch (java.lang.Exception ex) {
157:                 throw new IllegalStateException(ex.toString());
158:             }
159:         }
160:
161:         public java.lang.Object newInstance(java.lang.Object parent) {
162:             return null;
163:         }
164:     };
165:     handler = new org.exolab.castor.xml.handlers.EnumFieldHandler(DataSource.class, handler);
166:     desc.setImmutable(true);
167:     desc.setHandler(handler);
168:     desc.setRequired(true);
169:     desc.setMultivalued(false);
170:     addFieldDescriptor(desc);
171:
172:     //-- validation code for: _DS_ID
173:     fieldValidator = new org.exolab.castor.xml.FieldValidator();
174:     fieldValidator.setMinOccurs(1);
175:     { //-- local scope
176:     }
177:     desc.setValidator(fieldValidator);
```

```
178:
179:     }
180:     //-----/
181:     //- Methods -/
182:     //-----/
183:     /**
184:      * Method getAccessMode.
185:      *
186:      * @return the access mode specified for this class.
187:      */
188:     public org.exolab.castor.mapping.AccessMode getAccessMode() {
189:         return null;
190:     }
191:
192:     /**
193:      * Method getIdentity.
194:      *
195:      * @return the identity field, null if this class has no
196:      * identity.
197:      */
198:     public org.exolab.castor.mapping.FieldDescriptor getIdentity() {
199:         return _identity;
200:     }
201:
202:     /**
203:      * Method getJavaClass.
204:      *
205:      * @return the Java class represented by this descriptor.
206:      */
207:     public java.lang.Class getJavaClass() {
208:         return export.DataHeader.class;
209:     }
210:
211:     /**
212:      * Method getNamespacePrefix.
213:      *
214:      * @return the namespace prefix to use when marshaling as XML.
215:      */
216:     public java.lang.String getNamespacePrefix() {
217:         return _nsPrefix;
218:     }
219:
220:     /**
221:      * Method getNamespaceURI.
222:      *
```

```
223:      * @return the namespace URI used when marshaling and
224:      * unmarshaling as XML.
225:      */
226: public java.lang.String getNamespaceURI() {
227:     return _nsURI;
228: }
229:
230: /**
231:  * Method getValidator.
232:  *
233:  * @return a specific validator for the class described by this
234:  * ClassDescriptor.
235:  */
236: public org.exolab.castor.xml.TypeValidator getValidator() {
237:     return this;
238: }
239:
240: /**
241:  * Method getXMLName.
242:  *
243:  * @return the XML Name for the Class being described.
244:  */
245: public java.lang.String getXMLName() {
246:     return _xmlName;
247: }
248:
249: /**
250:  * Method isElementDefinition.
251:  *
252:  * @return true if XML schema definition of this Class is that
253:  * of a global
254:  * element or element with anonymous type definition.
255:  */
256: public boolean isElementDefinition() {
257:     return _elementDefinition;
258: }
259: }
```



```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      ECDataHeaderDescriptor.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: /**
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36: package export.descriptors;
37:
38: //-----/
39: //- Imported classes and packages -/
40: //-----/
41: import export.ECDataExport;
42:
43: /**
44:  * Class ECDataExportDescriptor.
45:  *
```

```
46:  * @version $Revision$ $Date$
47:  */
48: public class ECDataExportDescriptor extends org.exolab.castor.xml.util.XMLClassDescriptorImpl {
49:
50:
51:     //-----/
52:     //- Class/Member Variables -/
53:     //-----/
54:     /**
55:      * Field _elementDefinition.
56:      */
57:     private boolean _elementDefinition;
58:     /**
59:      * Field _nsPrefix.
60:      */
61:     private java.lang.String _nsPrefix;
62:     /**
63:      * Field _nsURI.
64:      */
65:     private java.lang.String _nsURI;
66:     /**
67:      * Field _xmlName.
68:      */
69:     private java.lang.String _xmlName;
70:     /**
71:      * Field _identity.
72:      */
73:     private org.exolab.castor.xml.XMLFieldDescriptor _identity;
74:
75:
76:     //-----/
77:     //- Constructors -/
78:     //-----/
79:     public ECDataExportDescriptor() {
80:         super();
81:         _xmlName = "ECDataExport";
82:         _elementDefinition = true;
83:
84:         //-- set grouping compositor
85:         setCompositorAsSequence();
86:         org.exolab.castor.xml.util.XMLFieldDescriptorImpl desc = null;
87:         org.exolab.castor.mapping.FieldHandler handler = null;
88:         org.exolab.castor.xml.FieldValidator fieldValidator = null;
89:         //-- initialize attribute descriptors
90:
```

BIRODatabaseManager/src/export/descriptors/ECDataExportDescriptor.java

```
91:         //-- initialize element descriptors
92:
93:         //-- _patient
94:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(export.Patient.class, "_patient", "Patient",
org.exolab.castor.xml.NodeType.Element);
95:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
96:
97:             public java.lang.Object getValue(java.lang.Object object)
98:                 throws IllegalStateException {
99:                 ECDataExport target = (ECDataExport) object;
100:                 return target.getPatient();
101:             }
102:
103:             public void setValue(java.lang.Object object, java.lang.Object value)
104:                 throws IllegalStateException, IllegalArgumentException {
105:                 try {
106:                     ECDataExport target = (ECDataExport) object;
107:                     target.setPatient((export.Patient) value);
108:                 } catch (java.lang.Exception ex) {
109:
110:                     throw new IllegalStateException(ex.toString());
111:                 }
112:             }
113:         }
114:
115:         public java.lang.Object newInstance(java.lang.Object parent) {
116:             return new export.Patient();
117:         }
118:     };
119:     desc.setHandler(handler);
120:     //desc.setRequired(true);
121:     desc.setMultivalued(true);
122:     addFieldDescriptor(desc);
123:
124:     //-- validation code for: _patientList
125:     fieldValidator = new org.exolab.castor.xml.FieldValidator();
126:     //fieldValidator.setMinOccurs(1);
127:     { //-- local scope
128:     }
129:     desc.setValidator(fieldValidator);
130:
131:     //-- _DataHeader
132:
133:     desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(export.DataHeader.class, "_dataHeader",
>DataHeader", org.exolab.castor.xml.NodeType.Element);
```

```
134:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
135:
136:             public java.lang.Object getValue(java.lang.Object object)
137:                 throws IllegalStateException {
138:                 ECDataExport target = (ECDataExport) object;
139:                 return target.getDataHeader();
140:             }
141:
142:             public void setValue(java.lang.Object object, java.lang.Object value)
143:                 throws IllegalStateException, IllegalArgumentException {
144:                 try {
145:                     ECDataExport target = (ECDataExport) object;
146:                     target.setDataHeader((export.DataHeader) value);
147:                 } catch (java.lang.Exception ex) {
148:                     System.out.println("DataHeaderSetValue");
149:                     throw new IllegalStateException(ex.toString());
150:                 }
151:             }
152:
153:             public java.lang.Object newInstance(java.lang.Object parent) {
154:                 return new export.DataHeader();
155:             }
156:         };
157:         desc.setHandler(handler);
158:         //desc.setRequired(true);
159:         desc.setMultivalued(true);
160:         addFieldDescriptor(desc);
161:
162:         //-- validation code for: _dataHeader
163:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
164:         //fieldValidator.setMinOccurs(1);
165:         { //-- local scope
166:         }
167:         desc.setValidator(fieldValidator);
168:
169:
170:     }
171:
172:
173:     //-----/
174:     //- Methods -/
175:     //-----/
176:     /**
177:      * Method getAccessMode.
178:      *
```

```
179:      * @return the access mode specified for this class.
180:      */
181:      public org.exolab.castor.mapping.AccessMode getAccessMode() {
182:          return null;
183:      }
184:
185:      /**
186:       * Method getIdentity.
187:       *
188:       * @return the identity field, null if this class has no
189:       * identity.
190:       */
191:      public org.exolab.castor.mapping.FieldDescriptor getIdentity() {
192:          return _identity;
193:      }
194:
195:      /**
196:       * Method getJavaClass.
197:       *
198:       * @return the Java class represented by this descriptor.
199:       */
200:      public java.lang.Class getJavaClass() {
201:          return export.ECDataExport.class;
202:      }
203:
204:      /**
205:       * Method getNamespacePrefix.
206:       *
207:       * @return the namespace prefix to use when marshaling as XML.
208:       */
209:      public java.lang.String getNamespacePrefix() {
210:          return _nsPrefix;
211:      }
212:
213:      /**
214:       * Method getNamespaceURI.
215:       *
216:       * @return the namespace URI used when marshaling and
217:       * unmarshaling as XML.
218:       */
219:      public java.lang.String getNamespaceURI() {
220:          return _nsURI;
221:      }
222:
223:      /**
```

```
224:      * Method getValidator.
225:      *
226:      * @return a specific validator for the class described by this
227:      * ClassDescriptor.
228:      */
229: public org.exolab.castor.xml.TypeValidator getValidator() {
230:     return this;
231: }
232:
233: /**
234:  * Method getXMLName.
235:  *
236:  * @return the XML Name for the Class being described.
237:  */
238: public java.lang.String getXMLName() {
239:     return _xmlName;
240: }
241:
242: /**
243:  * Method isElementDefinition.
244:  *
245:  * @return true if XML schema definition of this Class is that
246:  * of a global
247:  * element or element with anonymous type definition.
248:  */
249: public boolean isElementDefinition() {
250:     return _elementDefinition;
251: }
252: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      ECDataSourceExportDescriptor.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30:
31: /**
32:  * This class was automatically generated with
33:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
34:  * Schema.
35:  * $Id$
36:  */
37:
38: package export.descriptors;
39:
40: //-----/
41: //- Imported classes and packages -/
42: //-----/
43:
44: import export.ECDataSourceExport;
45:
```

```
46: /**
47:  * Class ECDataSourceExportDescriptor.
48:  *
49:  * @version $Revision$ $Date$
50:  */
51: public class ECDataSourceExportDescriptor extends org.exolab.castor.xml.util.XMLClassDescriptorImpl {
52:
53:
54:     //-----/
55:     //- Class/Member Variables -/
56:     //-----/
57:
58:     /**
59:      * Field _elementDefinition.
60:      */
61:     private boolean _elementDefinition;
62:
63:     /**
64:      * Field _nsPrefix.
65:      */
66:     private java.lang.String _nsPrefix;
67:
68:     /**
69:      * Field _nsURI.
70:      */
71:     private java.lang.String _nsURI;
72:
73:     /**
74:      * Field _xmlName.
75:      */
76:     private java.lang.String _xmlName;
77:
78:     /**
79:      * Field _identity.
80:      */
81:     private org.exolab.castor.xml.XMLFieldDescriptor _identity;
82:
83:
84:     //-----/
85:     //- Constructors -/
86:     //-----/
87:
88:     public ECDataSourceExportDescriptor() {
89:         super();
90:         _xmlName = "ECDataSourceExport";
```



```
91:         _elementDefinition = true;
92:
93:         //-- set grouping compositor
94:         setCompositorAsSequence();
95:         org.exolab.castor.xml.util.XMLFieldDescriptorImpl desc          = null;
96:         org.exolab.castor.mapping.FieldHandler handler              = null;
97:         org.exolab.castor.xml.FieldValidator fieldValidator = null;
98:         //-- initialize attribute descriptors
99:
100:        //-- initialize element descriptors
101:
102:        //-- _siteHeader
103:        desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(export.SiteHeader.class, "_siteHeader",
"SiteHeader", org.exolab.castor.xml.NodeType.Element);
104:        handler = new org.exolab.castor.xml.XMLFieldHandler() {
105:            public java.lang.Object getValue( java.lang.Object object )
106:                throws IllegalStateException
107:            {
108:                ECDataSourceExport target = (ECDataSourceExport) object;
109:                return target.getSiteHeader();
110:            }
111:            public void setValue( java.lang.Object object, java.lang.Object value)
112:                throws IllegalStateException, IllegalArgumentException
113:            {
114:                try {
115:                    ECDataSourceExport target = (ECDataSourceExport) object;
116:                    target.setSiteHeader( (export.SiteHeader) value);
117:                } catch (java.lang.Exception ex) {
118:                    throw new IllegalStateException(ex.toString());
119:                }
120:            }
121:            public java.lang.Object newInstance(java.lang.Object parent) {
122:                return new export.SiteHeader();
123:            }
124:        };
125:        desc.setHandler(handler);
126:        desc.setRequired(true);
127:        desc.setMultivalued(false);
128:        addFieldDescriptor(desc);
129:
130:        //-- validation code for: _siteHeader
131:        fieldValidator = new org.exolab.castor.xml.FieldValidator();
132:        fieldValidator.setMinOccurs(1);
133:        { //-- local scope
134:        }
```

```
135:         desc.setValidator(fieldValidator);
136:         //-- _siteProfile
137:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(export.SiteProfile.class, "_siteProfile",
"SiteProfile", org.exolab.castor.xml.NodeType.Element);
138:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
139:             public java.lang.Object getValue( java.lang.Object object )
140:                 throws IllegalStateException
141:             {
142:                 ECDataSourceExport target = (ECDataSourceExport) object;
143:                 return target.getSiteProfile();
144:             }
145:             public void setValue( java.lang.Object object, java.lang.Object value)
146:                 throws IllegalStateException, IllegalArgumentException
147:             {
148:                 try {
149:                     ECDataSourceExport target = (ECDataSourceExport) object;
150:                     target.setSiteProfile( (export.SiteProfile) value);
151:                 } catch (java.lang.Exception ex) {
152:                     throw new IllegalStateException(ex.toString());
153:                 }
154:             }
155:             public java.lang.Object newInstance(java.lang.Object parent) {
156:                 return new export.SiteProfile();
157:             }
158:         };
159:         desc.setHandler(handler);
160:         desc.setRequired(true);
161:         desc.setMultivalued(false);
162:         addFieldDescriptor(desc);
163:
164:         //-- validation code for: _siteProfile
165:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
166:         fieldValidator.setMinOccurs(1);
167:         { //-- local scope
168:         }
169:         desc.setValidator(fieldValidator);
170:         //-- _fieldExportProfilesList
171:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(export.FieldExportProfiles.class,
"_fieldExportProfilesList", "FieldExportProfiles", org.exolab.castor.xml.NodeType.Element);
172:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
173:             public java.lang.Object getValue( java.lang.Object object )
174:                 throws IllegalStateException
175:             {
176:                 ECDataSourceExport target = (ECDataSourceExport) object;
177:                 return target.getFieldExportProfiles();
```

```
178:     }
179:     public void setValue( java.lang.Object object, java.lang.Object value)
180:         throws IllegalStateException, IllegalArgumentException
181:     {
182:         try {
183:             ECDataSourceExport target = (ECDataSourceExport) object;
184:             target.addFieldExportProfiles( (export.FieldExportProfiles) value);
185:         } catch (java.lang.Exception ex) {
186:             throw new IllegalStateException(ex.toString());
187:         }
188:     }
189:     public void resetValue(Object object) throws IllegalStateException, IllegalArgumentException {
190:         try {
191:             ECDataSourceExport target = (ECDataSourceExport) object;
192:             target.removeAllFieldExportProfiles();
193:         } catch (java.lang.Exception ex) {
194:             throw new IllegalStateException(ex.toString());
195:         }
196:     }
197:     public java.lang.Object newInstance(java.lang.Object parent) {
198:         return new export.FieldExportProfiles();
199:     }
200: };
201: desc.setHandler(handler);
202: desc.setRequired(true);
203: desc.setMultivalued(true);
204: addFieldDescriptor(desc);
205:
206:     //-- validation code for: _fieldExportProfilesList
207:     fieldValidator = new org.exolab.castor.xml.FieldValidator();
208:     fieldValidator.setMinOccurs(1);
209:     { //-- local scope
210:     }
211:     desc.setValidator(fieldValidator);
212: }
213:
214:
215:     //-----/
216:     //- Methods -/
217:     //-----/
218:
219: /**
220:  * Method getAccessMode.
221:  *
222:  * @return the access mode specified for this class.
```

```
223:     */
224:     public org.exolab.castor.mapping.AccessMode getAccessMode(
225:     ) {
226:         return null;
227:     }
228:
229:     /**
230:      * Method getIdentity.
231:      *
232:      * @return the identity field, null if this class has no
233:      * identity.
234:      */
235:     public org.exolab.castor.mapping.FieldDescriptor getIdentity(
236:     ) {
237:         return _identity;
238:     }
239:
240:     /**
241:      * Method getJavaClass.
242:      *
243:      * @return the Java class represented by this descriptor.
244:      */
245:     public java.lang.Class getJavaClass(
246:     ) {
247:         return export.ECDataSourceExport.class;
248:     }
249:
250:     /**
251:      * Method getNamespacePrefix.
252:      *
253:      * @return the namespace prefix to use when marshaling as XML.
254:      */
255:     public java.lang.String getNamespacePrefix(
256:     ) {
257:         return _nsPrefix;
258:     }
259:
260:     /**
261:      * Method getNamespaceURI.
262:      *
263:      * @return the namespace URI used when marshaling and
264:      * unmarshaling as XML.
265:      */
266:     public java.lang.String getNamespaceURI(
267:     ) {
```

```
268:         return _nsURI;
269:     }
270:
271:     /**
272:      * Method getValidator.
273:      *
274:      * @return a specific validator for the class described by this
275:      * ClassDescriptor.
276:      */
277:     public org.exolab.castor.xml.TypeValidator getValidator(
278:     ) {
279:         return this;
280:     }
281:
282:     /**
283:      * Method getXMLName.
284:      *
285:      * @return the XML Name for the Class being described.
286:      */
287:     public java.lang.String getXMLName(
288:     ) {
289:         return _xmlName;
290:     }
291:
292:     /**
293:      * Method isElementDefinition.
294:      *
295:      * @return true if XML schema definition of this Class is that
296:      * of a global
297:      * element or element with anonymous type definition.
298:      */
299:     public boolean isElementDefinition(
300:     ) {
301:         return _elementDefinition;
302:     }
303:
304: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      EpisodeDataDescriptor.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: /*
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36:
37: package export.descriptors;
38:
39:  //-----/
40:  //- Imported classes and packages -/
41:  //-----/
42:
43: import export.EpisodeData;
44:
45: /**
```

```
46:  * Class EpisodeDataDescriptor.
47:  *
48:  * @version $Revision$ $Date$
49:  */
50: public class EpisodeDataDescriptor extends org.exolab.castor.xml.util.XMLClassDescriptorImpl {
51:
52:
53:     //-----/
54:     //- Class/Member Variables -/
55:     //-----/
56:
57:     /**
58:      * Field _elementDefinition.
59:      */
60:     private boolean _elementDefinition;
61:
62:     /**
63:      * Field _nsPrefix.
64:      */
65:     private java.lang.String _nsPrefix;
66:
67:     /**
68:      * Field _nsURI.
69:      */
70:     private java.lang.String _nsURI;
71:
72:     /**
73:      * Field _xmlName.
74:      */
75:     private java.lang.String _xmlName;
76:
77:     /**
78:      * Field _identity.
79:      */
80:     private org.exolab.castor.xml.XMLFieldDescriptor _identity;
81:
82:
83:     //-----/
84:     //- Constructors -/
85:     //-----/
86:
87:     public EpisodeDataDescriptor() {
88:         super();
89:         _xmlName = "EpisodeData";
90:         _elementDefinition = true;
```

```
91:
92:     //-- set grouping compositor
93:     setCompositorAsSequence();
94:     org.exolab.castor.xml.util.XMLFieldDescriptorImpl desc          = null;
95:     org.exolab.castor.mapping.FieldHandler handler                = null;
96:     org.exolab.castor.xml.FieldValidator fieldValidator           = null;
97:     //-- initialize attribute descriptors
98:
99:     //-- initialize element descriptors
100:
101:     //-- _episodeDate
102:     desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(org.exolab.castor.types.Date.class,
"episodeDate", "EpisodeDate", org.exolab.castor.xml.NodeType.Element);
103:     handler = new org.exolab.castor.xml.XMLFieldHandler() {
104:         public java.lang.Object getValue( java.lang.Object object )
105:             throws IllegalStateException
106:         {
107:             EpisodeData target = (EpisodeData) object;
108:             return target.getEpisodeDate();
109:         }
110:         public void setValue( java.lang.Object object, java.lang.Object value)
111:             throws IllegalStateException, IllegalArgumentException
112:         {
113:             try {
114:                 EpisodeData target = (EpisodeData) object;
115:                 target.setEpisodeDate( (org.exolab.castor.types.Date) value);
116:             } catch (java.lang.Exception ex) {
117:                 throw new IllegalStateException(ex.toString());
118:             }
119:         }
120:         public java.lang.Object newInstance(java.lang.Object parent) {
121:             return new org.exolab.castor.types.Date();
122:         }
123:     };
124:     desc.setHandler(handler);
125:     desc.setRequired(true);
126:     desc.setMultivalued(false);
127:     addFieldDescriptor(desc);
128:
129:     //-- validation code for: _episodeDate
130:     fieldValidator = new org.exolab.castor.xml.FieldValidator();
131:     fieldValidator.setMinOccurs(1);
132:     { //-- local scope
133:         org.exolab.castor.xml.validators.DateTimeValidator typeValidator;
134:         typeValidator = new org.exolab.castor.xml.validators.DateTimeValidator();
```



```
135:         fieldValidator.setValidator(typeValidator);
136:     }
137:     desc.setValidator(fieldValidator);
138:     //-- _dataList
139:     desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(export.Data.class, "_dataList", "Data",
org.exolab.castor.xml.NodeType.Element);
140:     handler = new org.exolab.castor.xml.XMLFieldHandler() {
141:         public java.lang.Object getValue( java.lang.Object object )
142:             throws IllegalStateException
143:         {
144:             EpisodeData target = (EpisodeData) object;
145:             return target.getData();
146:         }
147:         public void setValue( java.lang.Object object, java.lang.Object value)
148:             throws IllegalStateException, IllegalArgumentException
149:         {
150:             try {
151:                 EpisodeData target = (EpisodeData) object;
152:                 target.addData( (export.Data) value);
153:             } catch (java.lang.Exception ex) {
154:                 throw new IllegalStateException(ex.toString());
155:             }
156:         }
157:         public void resetValue(Object object) throws IllegalStateException, IllegalArgumentException {
158:             try {
159:                 EpisodeData target = (EpisodeData) object;
160:                 target.removeAllData();
161:             } catch (java.lang.Exception ex) {
162:                 throw new IllegalStateException(ex.toString());
163:             }
164:         }
165:         public java.lang.Object newInstance(java.lang.Object parent) {
166:             return new export.Data();
167:         }
168:     };
169:     desc.setHandler(handler);
170:     desc.setRequired(true);
171:     desc.setMultivalued(true);
172:     addFieldDescriptor(desc);
173:
174:     //-- validation code for: _dataList
175:     fieldValidator = new org.exolab.castor.xml.FieldValidator();
176:     fieldValidator.setMinOccurs(1);
177:     { //-- local scope
178:     }
```

```
179:         desc.setValidator(fieldValidator);
180:     }
181:
182:
183:     //-----/
184:     //- Methods -/
185:     //-----/
186:
187:     /**
188:      * Method getAccessMode.
189:      *
190:      * @return the access mode specified for this class.
191:      */
192:     public org.exolab.castor.mapping.AccessMode getAccessMode(
193:     ) {
194:         return null;
195:     }
196:
197:     /**
198:      * Method getIdentity.
199:      *
200:      * @return the identity field, null if this class has no
201:      * identity.
202:      */
203:     public org.exolab.castor.mapping.FieldDescriptor getIdentity(
204:     ) {
205:         return _identity;
206:     }
207:
208:     /**
209:      * Method getJavaClass.
210:      *
211:      * @return the Java class represented by this descriptor.
212:      */
213:     public java.lang.Class getJavaClass(
214:     ) {
215:         return export.EpisodeData.class;
216:     }
217:
218:     /**
219:      * Method getNamespacePrefix.
220:      *
221:      * @return the namespace prefix to use when marshaling as XML.
222:      */
223:     public java.lang.String getNamespacePrefix(
```

```
224:     ) {
225:         return _nsPrefix;
226:     }
227:
228:     /**
229:      * Method getNameSpaceURI.
230:      *
231:      * @return the namespace URI used when marshaling and
232:      * unmarshaling as XML.
233:      */
234:     public java.lang.String getNameSpaceURI(
235:     ) {
236:         return _nsURI;
237:     }
238:
239:     /**
240:      * Method getValidator.
241:      *
242:      * @return a specific validator for the class described by this
243:      * ClassDescriptor.
244:      */
245:     public org.exolab.castor.xml.TypeValidator getValidator(
246:     ) {
247:         return this;
248:     }
249:
250:     /**
251:      * Method getXMLName.
252:      *
253:      * @return the XML Name for the Class being described.
254:      */
255:     public java.lang.String getXMLName(
256:     ) {
257:         return _xmlName;
258:     }
259:
260:     /**
261:      * Method isElementDefinition.
262:      *
263:      * @return true if XML schema definition of this Class is that
264:      * of a global
265:      * element or element with anonymous type definition.
266:      */
267:     public boolean isElementDefinition(
268:     ) {
```

05/19/09
20:24:08

BIRODatabaseManager/src/export/descriptors/EpisodeDataDescriptor.java

7

```
269:         return _elementDefinition;
270:     }
271:
272: }
```

```
1:
2: /**
3:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
4:  *
5:  * File:      FieldExportProfilesDescriptor.java
6:  * Authors:   Pietro Palladino, Valentina Baglioni
7:  *
8:  * COPYRIGHT INFORMATION
9:  * This file is free software; you can redistribute it and/or modify
10: * it under the terms of the GNU General Public License as published by
11: * the Free Software Foundation; either version 2, or (at your option)
12: * any later version.
13: *
14: * This file is distributed in the hope that it will be useful,
15: * but WITHOUT ANY WARRANTY; without even the implied warranty of
16: * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
17: * GNU General Public License for more details.
18: *
19: * You should have received a copy of the GNU General Public License
20: * along with this file; see the file COPYING. If not, write to
21: * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
22: *
23: * In short: you may use this file any way you like, as long as you
24: * don't charge money for it, remove this notice, or hold anyone liable
25: * for its results.
26: *
27:
28: * GPL Copyright, The BIRO Project
29: *
30: **/
31: /*
32:  * This class was automatically generated with
33:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
34:  * Schema.
35:  * $Id$
36:  */
37:
38: package export.descriptors;
39:
40: //-----/
41: //- Imported classes and packages -/
42: //-----/
43:
44: import export.FieldExportProfiles;
45: import export.types.BIRODataSet;
```

```
46: import export.types.Ranking;
47:
48:
49: /**
50:  * Class FieldExportProfilesDescriptor.
51:  *
52:  * @version $Revision$ $Date$
53:  */
54: public class FieldExportProfilesDescriptor extends org.exolab.castor.xml.util.XMLClassDescriptorImpl {
55:
56:
57:     //-----/
58:     //- Class/Member Variables -/
59:     //-----/
60:
61:     /**
62:      * Field _elementDefinition.
63:      */
64:     private boolean _elementDefinition;
65:
66:     /**
67:      * Field _nsPrefix.
68:      */
69:     private java.lang.String _nsPrefix;
70:
71:     /**
72:      * Field _nsURI.
73:      */
74:     private java.lang.String _nsURI;
75:
76:     /**
77:      * Field _xmlName.
78:      */
79:     private java.lang.String _xmlName;
80:
81:     /**
82:      * Field _identity.
83:      */
84:     private org.exolab.castor.xml.XMLFieldDescriptor _identity;
85:
86:
87:     //-----/
88:     //- Constructors -/
89:     //-----/
90:
```

BIRODatabaseManager/src/export/descriptors/FieldExportProfilesDescriptor.java

```
91: public FieldExportProfilesDescriptor() {
92:     super();
93:     _xmlName = "FieldExportProfiles";
94:     _elementDefinition = true;
95:
96:     //-- set grouping compositor
97:     setCompositorAsSequence();
98:     org.exolab.castor.xml.util.XMLFieldDescriptorImpl desc = null;
99:     org.exolab.castor.mapping.FieldHandler handler = null;
100:    org.exolab.castor.xml.FieldValidator fieldValidator = null;
101:    //-- initialize attribute descriptors
102:
103:    //-- initialize element descriptors
104:
105:    //-- _fieldName
106:    desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(BIRODataSet.class, "_fieldName", "FieldName",
org.exolab.castor.xml.NodeType.Element);
107:    handler = new org.exolab.castor.xml.XMLFieldHandler() {
108:        @Override
109:        public java.lang.Object getValue( java.lang.Object object )
110:            throws IllegalStateException
111:        {
112:            FieldExportProfiles target = (FieldExportProfiles) object;
113:            return target.getFieldName();
114:        }
115:        @Override
116:        public void setValue( java.lang.Object object, java.lang.Object value)
117:            throws IllegalStateException, IllegalArgumentException
118:        {
119:            try {
120:                FieldExportProfiles target = (FieldExportProfiles) object;
121:                target.setFieldName( (BIRODataSet) value);
122:            } catch (java.lang.Exception ex) {
123:                throw new IllegalStateException(ex.toString());
124:            }
125:        }
126:        @Override
127:        public java.lang.Object newInstance(java.lang.Object parent) {
128:            return null;
129:        }
130:    };
131:    handler = new org.exolab.castor.xml.handlers.EnumFieldHandler(BIRODataSet.class, handler);
132:    desc.setImmutable(true);
133:    desc.setHandler(handler);
134:    desc.setRequired(true);
```

BIRODatabaseManager/src/export/descriptors/FieldExportProfilesDescriptor.java

```
135:         desc.setMultivalued(false);
136:         addFieldDescriptor(desc);
137:
138:         //-- validation code for: _fieldName
139:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
140:         fieldValidator.setMinOccurs(1);
141:         { //-- local scope
142:         }
143:         desc.setValidator(fieldValidator);
144:         //-- _dateStatusLastReviewed
145:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(org.exolab.castor.types.Date.class,
"_dateStatusLastReviewed", "DateStatusLastReviewed", org.exolab.castor.xml.NodeType.Element);
146:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
147:             @Override
148:             public java.lang.Object getValue( java.lang.Object object )
149:                 throws IllegalStateException
150:             {
151:                 FieldExportProfiles target = (FieldExportProfiles) object;
152:                 return target.getDateStatusLastReviewed();
153:             }
154:             @Override
155:             public void setValue( java.lang.Object object, java.lang.Object value)
156:                 throws IllegalStateException, IllegalArgumentException
157:             {
158:                 try {
159:                     FieldExportProfiles target = (FieldExportProfiles) object;
160:                     target.setDateStatusLastReviewed( (org.exolab.castor.types.Date) value);
161:                 } catch (java.lang.Exception ex) {
162:                     throw new IllegalStateException(ex.toString());
163:                 }
164:             }
165:             @Override
166:             public java.lang.Object newInstance(java.lang.Object parent) {
167:                 return new org.exolab.castor.types.Date();
168:             }
169:         };
170:         desc.setHandler(handler);
171:         desc.setRequired(true);
172:         desc.setMultivalued(false);
173:         addFieldDescriptor(desc);
174:
175:         //-- validation code for: _dateStatusLastReviewed
176:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
177:         fieldValidator.setMinOccurs(1);
178:         { //-- local scope
```


BIRODatabaseManager/src/export/descriptors/FieldExportProfilesDescriptor.java

```
179:         org.exolab.castor.xml.validators.DateTimeValidator typeValidator;
180:         typeValidator = new org.exolab.castor.xml.validators.DateTimeValidator();
181:         fieldValidator.setValidator(typeValidator);
182:     }
183:     desc.setValidator(fieldValidator);
184:     //-- _recorded
185:     desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.Boolean.TYPE, "_recorded",
"Recorded", org.exolab.castor.xml.NodeType.Element);
186:     handler = new org.exolab.castor.xml.XMLFieldHandler() {
187:         @Override
188:         public java.lang.Object getValue( java.lang.Object object )
189:             throws IllegalStateException
190:         {
191:             FieldExportProfiles target = (FieldExportProfiles) object;
192:             if (!target.hasRecorded()) { return null; }
193:             return (target.getRecorded() ? java.lang.Boolean.TRUE : java.lang.Boolean.FALSE);
194:         }
195:         @Override
196:         public void setValue( java.lang.Object object, java.lang.Object value)
197:             throws IllegalStateException, IllegalArgumentException
198:         {
199:             try {
200:                 FieldExportProfiles target = (FieldExportProfiles) object;
201:                 // ignore null values for non optional primitives
202:                 if (value == null) { return; }
203:
204:                 target.setRecorded( ((java.lang.Boolean) value).booleanValue());
205:             } catch (java.lang.Exception ex) {
206:                 throw new IllegalStateException(ex.toString());
207:             }
208:         }
209:         @Override
210:         public java.lang.Object newInstance(java.lang.Object parent) {
211:             return null;
212:         }
213:     };
214:     desc.setHandler(handler);
215:     desc.setRequired(true);
216:     desc.setMultivalued(false);
217:     addFieldDescriptor(desc);
218:
219:     //-- validation code for: _recorded
220:     fieldValidator = new org.exolab.castor.xml.FieldValidator();
221:     fieldValidator.setMinOccurs(1);
222:     { //-- local scope
```

BIRODatabaseManager/src/export/descriptors/FieldExportProfilesDescriptor.java

```
223:         org.exolab.castor.xml.validators.BooleanValidator typeValidator;
224:         typeValidator = new org.exolab.castor.xml.validators.BooleanValidator();
225:         fieldValidator.setValidator(typeValidator);
226:     }
227:     desc.setValidator(fieldValidator);
228:     //-- _consistency
229:     desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(Ranking.class, "_consistency", "Consistency",
org.exolab.castor.xml.NodeType.Element);
230:     handler = new org.exolab.castor.xml.XMLFieldHandler() {
231:         @Override
232:         public java.lang.Object getValue( java.lang.Object object )
233:             throws IllegalStateException
234:         {
235:             FieldExportProfiles target = (FieldExportProfiles) object;
236:             return target.getConsistency();
237:         }
238:         @Override
239:         public void setValue( java.lang.Object object, java.lang.Object value)
240:             throws IllegalStateException, IllegalArgumentException
241:         {
242:             try {
243:                 FieldExportProfiles target = (FieldExportProfiles) object;
244:                 target.setConsistency( (Ranking) value);
245:             } catch (java.lang.Exception ex) {
246:                 throw new IllegalStateException(ex.toString());
247:             }
248:         }
249:         @Override
250:         public java.lang.Object newInstance(java.lang.Object parent) {
251:             return null;
252:         }
253:     };
254:     handler = new org.exolab.castor.xml.handlers.EnumFieldHandler(Ranking.class, handler);
255:     desc.setImmutable(true);
256:     desc.setHandler(handler);
257:     desc.setRequired(true);
258:     desc.setMultivalued(false);
259:     addFieldDescriptor(desc);
260:
261:     //-- validation code for: _consistency
262:     fieldValidator = new org.exolab.castor.xml.FieldValidator();
263:     fieldValidator.setMinOccurs(1);
264:     { //-- local scope
265:     }
266:     desc.setValidator(fieldValidator);
```

```
267:         //-- _completeness
268:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl( java.lang.Long.TYPE, "_completeness",
"Completeness", org.exolab.castor.xml.NodeType.Element);
269:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
270:             @Override
271:             public java.lang.Object getValue( java.lang.Object object )
272:                 throws IllegalStateException
273:             {
274:                 FieldExportProfiles target = (FieldExportProfiles) object;
275:                 if (!target.hasCompleteness()) { return null; }
276:                 return new java.lang.Long(target.getCompleteness());
277:             }
278:             @Override
279:             public void setValue( java.lang.Object object, java.lang.Object value)
280:                 throws IllegalStateException, IllegalArgumentException
281:             {
282:                 try {
283:                     FieldExportProfiles target = (FieldExportProfiles) object;
284:                     // ignore null values for non optional primitives
285:                     if (value == null) { return; }
286:
287:                     target.setCompleteness( ((java.lang.Long) value).longValue());
288:                 } catch (java.lang.Exception ex) {
289:                     throw new IllegalStateException(ex.toString());
290:                 }
291:             }
292:             @Override
293:             public java.lang.Object newInstance(java.lang.Object parent) {
294:                 return null;
295:             }
296:         };
297:         desc.setHandler(handler);
298:         desc.setRequired(true);
299:         desc.setMultivalued(false);
300:         addFieldDescriptor(desc);
301:
302:         //-- validation code for: _completeness
303:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
304:         //fieldValidator.setMinOccurs(1);
305:         { //-- local scope
306:             org.exolab.castor.xml.validators.LongValidator typeValidator;
307:             typeValidator = new org.exolab.castor.xml.validators.LongValidator();
308:             fieldValidator.setValidator(typeValidator);
309:             // typeValidator.setMinInclusive(1L);
310:         }
```

```
311:         desc.setValidator(fieldValidator);
312:         //-- _mandatory
313:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.Boolean.TYPE, "_mandatory",
"Mandatory", org.exolab.castor.xml.NodeType.Element);
314:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
315:             @Override
316:             public java.lang.Object getValue( java.lang.Object object )
317:                 throws IllegalStateException
318:             {
319:                 FieldExportProfiles target = (FieldExportProfiles) object;
320:                 if (!target.hasMandatory()) { return null; }
321:                 return (target.getMandatory() ? java.lang.Boolean.TRUE : java.lang.Boolean.FALSE);
322:             }
323:             @Override
324:             public void setValue( java.lang.Object object, java.lang.Object value)
325:                 throws IllegalStateException, IllegalArgumentException
326:             {
327:                 try {
328:                     FieldExportProfiles target = (FieldExportProfiles) object;
329:                     // ignore null values for non optional primitives
330:                     if (value == null) { return; }
331:
332:                     target.setMandatory( ((java.lang.Boolean) value).booleanValue());
333:                 } catch (java.lang.Exception ex) {
334:                     throw new IllegalStateException(ex.toString());
335:                 }
336:             }
337:             @Override
338:             public java.lang.Object newInstance(java.lang.Object parent) {
339:                 return null;
340:             }
341:         };
342:         desc.setHandler(handler);
343:         desc.setRequired(true);
344:         desc.setMultivalued(false);
345:         addFieldDescriptor(desc);
346:
347:         //-- validation code for: _mandatory
348:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
349:         fieldValidator.setMinOccurs(1);
350:         { //-- local scope
351:             org.exolab.castor.xml.validators.BooleanValidator typeValidator;
352:             typeValidator = new org.exolab.castor.xml.validators.BooleanValidator();
353:             fieldValidator.setValidator(typeValidator);
354:         }
```

```
355:         desc.setValidator(fieldValidator);
356:         //-- _routine
357:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.Boolean.TYPE, "_routine", "Routine"
, org.exolab.castor.xml.NodeType.Element);
358:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
359:             @Override
360:             public java.lang.Object getValue( java.lang.Object object )
361:                 throws IllegalStateException
362:             {
363:                 FieldExportProfiles target = (FieldExportProfiles) object;
364:                 if (!target.hasRoutine()) { return null; }
365:                 return (target.getRoutine() ? java.lang.Boolean.TRUE : java.lang.Boolean.FALSE);
366:             }
367:             @Override
368:             public void setValue( java.lang.Object object, java.lang.Object value)
369:                 throws IllegalStateException, IllegalArgumentException
370:             {
371:                 try {
372:                     FieldExportProfiles target = (FieldExportProfiles) object;
373:                     // ignore null values for non optional primitives
374:                     if (value == null) { return; }
375:
376:                     target.setRoutine( ((java.lang.Boolean) value).booleanValue());
377:                 } catch (java.lang.Exception ex) {
378:                     throw new IllegalStateException(ex.toString());
379:                 }
380:             }
381:             @Override
382:             public java.lang.Object newInstance(java.lang.Object parent) {
383:                 return null;
384:             }
385:         };
386:         desc.setHandler(handler);
387:         desc.setRequired(true);
388:         desc.setMultivalued(false);
389:         addFieldDescriptor(desc);
390:
391:         //-- validation code for: _routine
392:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
393:         fieldValidator.setMinOccurs(1);
394:         { //-- local scope
395:             org.exolab.castor.xml.validators.BooleanValidator typeValidator;
396:             typeValidator = new org.exolab.castor.xml.validators.BooleanValidator();
397:             fieldValidator.setValidator(typeValidator);
398:         }
```

```
399:         desc.setValidator(fieldValidator);
400:         //-- _qualityScore
401:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(Ranking.class, "_qualityScore", "QualityScore"
, org.exolab.castor.xml.NodeType.Element);
402:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
403:             @Override
404:             public java.lang.Object getValue( java.lang.Object object )
405:                 throws IllegalStateException
406:             {
407:                 FieldExportProfiles target = (FieldExportProfiles) object;
408:                 return target.getQualityScore();
409:             }
410:             @Override
411:             public void setValue( java.lang.Object object, java.lang.Object value)
412:                 throws IllegalStateException, IllegalArgumentException
413:             {
414:                 try {
415:                     FieldExportProfiles target = (FieldExportProfiles) object;
416:                     target.setQualityScore( (Ranking) value);
417:                 } catch (java.lang.Exception ex) {
418:                     throw new IllegalStateException(ex.toString());
419:                 }
420:             }
421:             @Override
422:             public java.lang.Object newInstance(java.lang.Object parent) {
423:                 return null;
424:             }
425:         };
426:         handler = new org.exolab.castor.xml.handlers.EnumFieldHandler(Ranking.class, handler);
427:         desc.setImmutable(true);
428:         desc.setHandler(handler);
429:         desc.setRequired(true);
430:         desc.setMultivalued(false);
431:         addFieldDescriptor(desc);
432:
433:         //-- validation code for: _qualityScore
434:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
435:         fieldValidator.setMinOccurs(1);
436:         { //-- local scope
437:         }
438:         desc.setValidator(fieldValidator);
439:         //-- _fieldExportComments
440:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.String.class, "_fieldExportComments"
, "FieldExportComments", org.exolab.castor.xml.NodeType.Element);
441:         desc.setImmutable(true);
```

```
442:     handler = new org.exolab.castor.xml.XMLFieldHandler() {
443:         @Override
444:         public java.lang.Object getValue( java.lang.Object object )
445:             throws IllegalStateException
446:         {
447:             FieldExportProfiles target = (FieldExportProfiles) object;
448:             return target.getFieldExportComments();
449:         }
450:         @Override
451:         public void setValue( java.lang.Object object, java.lang.Object value)
452:             throws IllegalStateException, IllegalArgumentException
453:         {
454:             try {
455:                 FieldExportProfiles target = (FieldExportProfiles) object;
456:                 target.setFieldExportComments( (java.lang.String) value);
457:             } catch (java.lang.Exception ex) {
458:                 throw new IllegalStateException(ex.toString());
459:             }
460:         }
461:         @Override
462:         public java.lang.Object newInstance(java.lang.Object parent) {
463:             return null;
464:         }
465:     };
466:     desc.setHandler(handler);
467:     desc.setMultivalued(false);
468:     addFieldDescriptor(desc);
469:
470:     //-- validation code for: _fieldExportComments
471:     fieldValidator = new org.exolab.castor.xml.FieldValidator();
472:     { //-- local scope
473:         org.exolab.castor.xml.validators.StringValidator typeValidator;
474:         typeValidator = new org.exolab.castor.xml.validators.StringValidator();
475:         fieldValidator.setValidator(typeValidator);
476:         typeValidator.setWhiteSpace("preserve");
477:     }
478:     desc.setValidator(fieldValidator);
479: }
480:
481:
482:     //-----/
483:     //- Methods -/
484:     //-----/
485:
486: /**
```

```
487:      * Method getAccessMode.
488:      *
489:      * @return the access mode specified for this class.
490:      */
491: public org.exolab.castor.mapping.AccessMode getAccessMode(
492: ) {
493:     return null;
494: }
495:
496: /**
497:  * Method getIdentity.
498:  *
499:  * @return the identity field, null if this class has no
500:  * identity.
501:  */
502: public org.exolab.castor.mapping.FieldDescriptor getIdentity(
503: ) {
504:     return _identity;
505: }
506:
507: /**
508:  * Method getJavaClass.
509:  *
510:  * @return the Java class represented by this descriptor.
511:  */
512: public java.lang.Class getJavaClass(
513: ) {
514:     return export.FieldExportProfiles.class;
515: }
516:
517: /**
518:  * Method getNameSpacePrefix.
519:  *
520:  * @return the namespace prefix to use when marshaling as XML.
521:  */
522: public java.lang.String getNameSpacePrefix(
523: ) {
524:     return _nsPrefix;
525: }
526:
527: /**
528:  * Method getNameSpaceURI.
529:  *
530:  * @return the namespace URI used when marshaling and
531:  * unmarshaling as XML.
```



```
532:     */
533:     public java.lang.String getNameSpaceURI(
534:     ) {
535:         return _nsURI;
536:     }
537:
538:     /**
539:      * Method getValidator.
540:      *
541:      * @return a specific validator for the class described by this
542:      * ClassDescriptor.
543:      */
544:     public org.exolab.castor.xml.TypeValidator getValidator(
545:     ) {
546:         return this;
547:     }
548:
549:     /**
550:      * Method getXMLName.
551:      *
552:      * @return the XML Name for the Class being described.
553:      */
554:     public java.lang.String getXMLName(
555:     ) {
556:         return _xmlName;
557:     }
558:
559:     /**
560:      * Method isElementDefinition.
561:      *
562:      * @return true if XML schema definition of this Class is that
563:      * of a global
564:      * element or element with anonymous type definition.
565:      */
566:     public boolean isElementDefinition(
567:     ) {
568:         return _elementDefinition;
569:     }
570:
571: }
```

```
1:
2: /**
3:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
4:  *
5:  * File:      PatientDescriptor.java
6:  * Authors:   Pietro Palladino, Valentina Baglioni
7:  *
8:  * COPYRIGHT INFORMATION
9:  * This file is free software; you can redistribute it and/or modify
10: * it under the terms of the GNU General Public License as published by
11: * the Free Software Foundation; either version 2, or (at your option)
12: * any later version.
13: *
14: * This file is distributed in the hope that it will be useful,
15: * but WITHOUT ANY WARRANTY; without even the implied warranty of
16: * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
17: * GNU General Public License for more details.
18: *
19: * You should have received a copy of the GNU General Public License
20: * along with this file; see the file COPYING. If not, write to
21: * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
22: *
23: * In short: you may use this file any way you like, as long as you
24: * don't charge money for it, remove this notice, or hold anyone liable
25: * for its results.
26: *
27:
28: * GPL Copyright, The BIRO Project
29: *
30: **/
31: /*
32:  * This class was automatically generated with
33:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
34:  * Schema.
35:  * $Id$
36:  */
37: package export.descriptors;
38:
39: //-----/
40: //- Imported classes and packages -/
41: //-----/
42: import export.ActivityData;
43: import export.EpisodeData;
44: import export.Patient;
45: import export.Profile;
```

```
46:
47: /**
48:  * Class PatientDescriptor.
49:  *
50:  * @version $Revision$ $Date$
51:  */
52: public class PatientDescriptor extends org.exolab.castor.xml.util.XMLClassDescriptorImpl {
53:
54:
55:     //-----/
56:     //- Class/Member Variables -/
57:     //-----/
58:     /**
59:      * Field _elementDefinition.
60:      */
61:     private boolean _elementDefinition;
62:     /**
63:      * Field _nsPrefix.
64:      */
65:     private java.lang.String _nsPrefix;
66:     /**
67:      * Field _nsURI.
68:      */
69:     private java.lang.String _nsURI;
70:     /**
71:      * Field _xmlName.
72:      */
73:     private java.lang.String _xmlName;
74:     /**
75:      * Field _identity.
76:      */
77:     private org.exolab.castor.xml.XMLFieldDescriptor _identity;
78:
79:
80:     //-----/
81:     //- Constructors -/
82:     //-----/
83:     public PatientDescriptor() {
84:         super();
85:         _xmlName = "Patient";
86:         _elementDefinition = true;
87:
88:         //-- set grouping compositor
89:         setCompositorAsSequence();
90:         org.exolab.castor.xml.util.XMLFieldDescriptorImpl desc = null;
```

BIRODatabaseManager/src/export/descriptors/PatientDescriptor.java

```
91:         org.exolab.castor.mapping.FieldHandler handler = null;
92:         org.exolab.castor.xml.FieldValidator fieldValidator = null;
93:         //-- initialize attribute descriptors
94:
95:         //-- initialize element descriptors
96:
97:         //-- _profileList
98:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(Profile.class, "_profileList", "Profile",
org.exolab.castor.xml.NodeType.Element);
99:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
100:
101:             public java.lang.Object getValue(java.lang.Object object)
102:                 throws IllegalStateException {
103:                 Patient target = (Patient) object;
104:                 return target.getProfile();
105:             }
106:
107:             public void setValue(java.lang.Object object, java.lang.Object value)
108:                 throws IllegalStateException, IllegalArgumentException {
109:                 try {
110:                     Patient target = (Patient) object;
111:                     target.addProfile((Profile) value);
112:                 } catch (java.lang.Exception ex) {
113:                     throw new IllegalStateException(ex.toString());
114:                 }
115:             }
116:
117:             public void resetValue(Object object) throws IllegalStateException, IllegalArgumentException {
118:                 try {
119:                     Patient target = (Patient) object;
120:                     target.removeAllProfile();
121:                 } catch (java.lang.Exception ex) {
122:                     throw new IllegalStateException(ex.toString());
123:                 }
124:             }
125:
126:             public java.lang.Object newInstance(java.lang.Object parent) {
127:                 return new export.Profile();
128:             }
129:         };
130:         desc.setHandler(handler);
131:         desc.setRequired(true);
132:         desc.setMultivalued(true);
133:         addFieldDescriptor(desc);
134:
```

BIRODatabaseManager/src/export/descriptors/PatientDescriptor.java

```
135:         //-- validation code for: _profileList
136:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
137:         fieldValidator.setMinOccurs(1);
138:         { //-- local scope
139:         }
140:         desc.setValidator(fieldValidator);
141:
142:         //-- _activityDataList
143:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(ActivityData.class, "_activityDataList",
"ActivityData", org.exolab.castor.xml.NodeType.Element);
144:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
145:
146:             public java.lang.Object getValue(java.lang.Object object)
147:                 throws IllegalStateException {
148:                 Patient target = (Patient) object;
149:                 return target.getActivityData();
150:             }
151:
152:             public void setValue(java.lang.Object object, java.lang.Object value)
153:                 throws IllegalStateException, IllegalArgumentException {
154:                 try {
155:                     Patient target = (Patient) object;
156:                     target.addActivityData((ActivityData) value);
157:                 } catch (java.lang.Exception ex) {
158:                     throw new IllegalStateException(ex.toString());
159:                 }
160:             }
161:
162:             public void resetValue(Object object) throws IllegalStateException, IllegalArgumentException {
163:                 try {
164:                     Patient target = (Patient) object;
165:                     target.removeAllActivityData();
166:                 } catch (java.lang.Exception ex) {
167:                     throw new IllegalStateException(ex.toString());
168:                 }
169:             }
170:
171:             public java.lang.Object newInstance(java.lang.Object parent) {
172:                 return new ActivityData();
173:             }
174:         };
175:         desc.setHandler(handler);
176:         desc.setMultivalued(true);
177:         addFieldDescriptor(desc);
178:
```

BIRODatabaseManager/src/export/descriptors/PatientDescriptor.java

```
179:      //-- validation code for: _activityDataList
180:      fieldValidator = new org.exolab.castor.xml.FieldValidator();
181:      fieldValidator.setMinOccurs(0);
182:      { //-- local scope
183:      }
184:      desc.setValidator(fieldValidator);
185:
186:
187:      //-- _episodeDataList
188:      desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(EpisodeData.class, "_episodeDataList",
"EpisodeData", org.exolab.castor.xml.NodeType.Element);
189:      handler = new org.exolab.castor.xml.XMLFieldHandler() {
190:
191:          public java.lang.Object getValue(java.lang.Object object)
192:              throws IllegalStateException {
193:              Patient target = (Patient) object;
194:              return target.getEpisodeData();
195:          }
196:
197:          public void setValue(java.lang.Object object, java.lang.Object value)
198:              throws IllegalStateException, IllegalArgumentException {
199:              try {
200:                  Patient target = (Patient) object;
201:                  target.addEpisodeData((EpisodeData) value);
202:              } catch (java.lang.Exception ex) {
203:                  throw new IllegalStateException(ex.toString());
204:              }
205:          }
206:
207:          public void resetValue(Object object) throws IllegalStateException, IllegalArgumentException {
208:              try {
209:                  Patient target = (Patient) object;
210:                  target.removeAllEpisodeData();
211:              } catch (java.lang.Exception ex) {
212:                  throw new IllegalStateException(ex.toString());
213:              }
214:          }
215:
216:          public java.lang.Object newInstance(java.lang.Object parent) {
217:              return new export.EpisodeData();
218:          }
219:      };
220:      desc.setHandler(handler);
221:      //desc.setRequired(true);
222:      desc.setMultivalued(true);
```

```
223:         addFieldDescriptor(desc);
224:
225:         //-- validation code for: _episodeDataList
226:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
227:         fieldValidator.setMinOccurs(1);
228:         { //-- local scope
229:         }
230:         desc.setValidator(fieldValidator);
231:     }
232:
233:
234:     //-----/
235:     //- Methods -/
236:     //-----/
237:     /**
238:      * Method getAccessMode.
239:      *
240:      * @return the access mode specified for this class.
241:      */
242:     public org.exolab.castor.mapping.AccessMode getAccessMode() {
243:         return null;
244:     }
245:
246:     /**
247:      * Method getIdentity.
248:      *
249:      * @return the identity field, null if this class has no
250:      * identity.
251:      */
252:     public org.exolab.castor.mapping.FieldDescriptor getIdentity() {
253:         return _identity;
254:     }
255:
256:     /**
257:      * Method getJavaClass.
258:      *
259:      * @return the Java class represented by this descriptor.
260:      */
261:     public java.lang.Class getJavaClass() {
262:         return export.Patient.class;
263:     }
264:
265:     /**
266:      * Method getNameSpacePrefix.
267:      *
```

```
268:      * @return the namespace prefix to use when marshaling as XML.
269:      */
270: public java.lang.String getNamespacePrefix() {
271:     return _nsPrefix;
272: }
273:
274: /**
275:  * Method getNamespaceURI.
276:  *
277:  * @return the namespace URI used when marshaling and
278:  * unmarshaling as XML.
279:  */
280: public java.lang.String getNamespaceURI() {
281:     return _nsURI;
282: }
283:
284: /**
285:  * Method getValidator.
286:  *
287:  * @return a specific validator for the class described by this
288:  * ClassDescriptor.
289:  */
290: public org.exolab.castor.xml.TypeValidator getValidator() {
291:     return this;
292: }
293:
294: /**
295:  * Method getXMLName.
296:  *
297:  * @return the XML Name for the Class being described.
298:  */
299: public java.lang.String getXMLName() {
300:     return _xmlName;
301: }
302:
303: /**
304:  * Method isElementDefinition.
305:  *
306:  * @return true if XML schema definition of this Class is that
307:  * of a global
308:  * element or element with anonymous type definition.
309:  */
310: public boolean isElementDefinition() {
311:     return _elementDefinition;
312: }
```


05/19/09
20:25:28

BIRODatabaseManager/src/export/descriptors/PatientDescriptor.java

8

313: }

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      ProfileDescriptor.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30:
31: /**
32:  * This class was automatically generated with
33:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
34:  * Schema.
35:  * $Id$
36:  */
37:
38: package export.descriptors;
39:
40:  //-----/
41:  //- Imported classes and packages -/
42:  //-----/
43:
44: import export.Profile;
45: import export.types.BIRODataSet;
```

```
46:
47: /**
48:  * Class ProfileDescriptor.
49:  *
50:  * @version $Revision$ $Date$
51:  */
52: public class ProfileDescriptor extends org.exolab.castor.xml.util.XMLClassDescriptorImpl {
53:
54:
55:     //-----/
56:     //- Class/Member Variables -/
57:     //-----/
58:
59:     /**
60:      * Field _elementDefinition.
61:      */
62:     private boolean _elementDefinition;
63:
64:     /**
65:      * Field _nsPrefix.
66:      */
67:     private java.lang.String _nsPrefix;
68:
69:     /**
70:      * Field _nsURI.
71:      */
72:     private java.lang.String _nsURI;
73:
74:     /**
75:      * Field _xmlName.
76:      */
77:     private java.lang.String _xmlName;
78:
79:     /**
80:      * Field _identity.
81:      */
82:     private org.exolab.castor.xml.XMLFieldDescriptor _identity;
83:
84:
85:     //-----/
86:     //- Constructors -/
87:     //-----/
88:
89:     public ProfileDescriptor() {
90:         super();
```

```
91:         _xmlName = "Profile";
92:         _elementDefinition = true;
93:
94:         //-- set grouping compositor
95:         setCompositorAsSequence();
96:         org.exolab.castor.xml.util.XMLFieldDescriptorImpl desc          = null;
97:         org.exolab.castor.mapping.FieldHandler handler              = null;
98:         org.exolab.castor.xml.FieldValidator fieldValidator = null;
99:         //-- initialize attribute descriptors
100:
101:         //-- initialize element descriptors
102:
103:         //-- _profileFieldName
104:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(BIRODataSet.class, "_profileFieldName",
"ProfileFieldName", org.exolab.castor.xml.NodeType.Element);
105:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
106:             public java.lang.Object getValue( java.lang.Object object )
107:                 throws IllegalStateException
108:             {
109:                 Profile target = (Profile) object;
110:                 return target.getProfileFieldName();
111:             }
112:             public void setValue( java.lang.Object object, java.lang.Object value)
113:                 throws IllegalStateException, IllegalArgumentException
114:             {
115:                 try {
116:                     Profile target = (Profile) object;
117:                     target.setProfileFieldName( (BIRODataSet) value);
118:                 } catch (java.lang.Exception ex) {
119:                     throw new IllegalStateException(ex.toString());
120:                 }
121:             }
122:             public java.lang.Object newInstance(java.lang.Object parent) {
123:                 return null;
124:             }
125:         };
126:         handler = new org.exolab.castor.xml.handlers.EnumFieldHandler(BIRODataSet.class, handler);
127:         desc.setImmutable(true);
128:         desc.setHandler(handler);
129:         desc.setRequired(true);
130:         desc.setMultivalued(false);
131:         addFieldDescriptor(desc);
132:
133:         //-- validation code for: _profileFieldName
134:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
```

```
135:         fieldValidator.setMinOccurs(1);
136:         { //-- local scope
137:         }
138:         desc.setValidator(fieldValidator);
139:         //-- _profileFieldValue
140:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.String.class, "_profileFieldValue",
"ProfileFieldValue", org.exolab.castor.xml.NodeType.Element);
141:         desc.setImmutable(true);
142:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
143:             public java.lang.Object getValue( java.lang.Object object )
144:                 throws IllegalStateException
145:             {
146:                 Profile target = (Profile) object;
147:                 return target.getProfileFieldValue();
148:             }
149:             public void setValue( java.lang.Object object, java.lang.Object value)
150:                 throws IllegalStateException, IllegalArgumentException
151:             {
152:                 try {
153:                     Profile target = (Profile) object;
154:                     target.setProfileFieldValue( (java.lang.String) value);
155:                 } catch (java.lang.Exception ex) {
156:                     throw new IllegalStateException(ex.toString());
157:                 }
158:             }
159:             public java.lang.Object newInstance(java.lang.Object parent) {
160:                 return null;
161:             }
162:         };
163:         desc.setHandler(handler);
164:         desc.setRequired(true);
165:         desc.setMultivalued(false);
166:         addFieldDescriptor(desc);
167:
168:         //-- validation code for: _profileFieldValue
169:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
170:         fieldValidator.setMinOccurs(1);
171:         { //-- local scope
172:             org.exolab.castor.xml.validators.StringValidator typeValidator;
173:             typeValidator = new org.exolab.castor.xml.validators.StringValidator();
174:             fieldValidator.setValidator(typeValidator);
175:             typeValidator.setWhiteSpace("preserve");
176:         }
177:         desc.setValidator(fieldValidator);
178:     }
```

```
179:
180:
181:     //-----/
182:     //- Methods -/
183:     //-----/
184:
185:     /**
186:      * Method getAccessMode.
187:      *
188:      * @return the access mode specified for this class.
189:      */
190:     public org.exolab.castor.mapping.AccessMode getAccessMode(
191:     ) {
192:         return null;
193:     }
194:
195:     /**
196:      * Method getIdentity.
197:      *
198:      * @return the identity field, null if this class has no
199:      * identity.
200:      */
201:     public org.exolab.castor.mapping.FieldDescriptor getIdentity(
202:     ) {
203:         return _identity;
204:     }
205:
206:     /**
207:      * Method getJavaClass.
208:      *
209:      * @return the Java class represented by this descriptor.
210:      */
211:     public java.lang.Class getJavaClass(
212:     ) {
213:         return export.Profile.class;
214:     }
215:
216:     /**
217:      * Method getNameSpacePrefix.
218:      *
219:      * @return the namespace prefix to use when marshaling as XML.
220:      */
221:     public java.lang.String getNameSpacePrefix(
222:     ) {
223:         return _nsPrefix;
```

```
224:     }
225:
226:     /**
227:      * Method getNameSpaceURI.
228:      *
229:      * @return the namespace URI used when marshaling and
230:      * unmarshaling as XML.
231:      */
232:     public java.lang.String getNameSpaceURI(
233:     ) {
234:         return _nsURI;
235:     }
236:
237:     /**
238:      * Method getValidator.
239:      *
240:      * @return a specific validator for the class described by this
241:      * ClassDescriptor.
242:      */
243:     public org.exolab.castor.xml.TypeValidator getValidator(
244:     ) {
245:         return this;
246:     }
247:
248:     /**
249:      * Method getXMLName.
250:      *
251:      * @return the XML Name for the Class being described.
252:      */
253:     public java.lang.String getXMLName(
254:     ) {
255:         return _xmlName;
256:     }
257:
258:     /**
259:      * Method isElementDefinition.
260:      *
261:      * @return true if XML schema definition of this Class is that
262:      * of a global
263:      * element or element with anonymous type definition.
264:      */
265:     public boolean isElementDefinition(
266:     ) {
267:         return _elementDefinition;
268:     }
```

05/19/09
20:25:26

BIRODatabaseManager/src/export/descriptors/ProfileDescriptor.java

7

```
269:
270: }
```



```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      SiteHeaderDescriptor.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30:
31: /**
32:  * This class was automatically generated with
33:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
34:  * Schema.
35:  * $Id$
36:  */
37: package export.descriptors;
38:
39: //---------------------------------/
40: //- Imported classes and packages -/
41: //---------------------------------/
42: import export.SiteHeader;
43: import export.types.DataSource;
44:
45: /**
```

```
46:  * Class SiteHeaderDescriptor.
47:  *
48:  * @version $Revision$ $Date$
49:  */
50: public class SiteHeaderDescriptor extends org.exolab.castor.xml.util.XMLClassDescriptorImpl {
51:
52:
53:     //-----/
54:     //- Class/Member Variables -/
55:     //-----/
56:     /**
57:      * Field _elementDefinition.
58:      */
59:     private boolean _elementDefinition;
60:     /**
61:      * Field _nsPrefix.
62:      */
63:     private java.lang.String _nsPrefix;
64:     /**
65:      * Field _nsURI.
66:      */
67:     private java.lang.String _nsURI;
68:     /**
69:      * Field _xmlName.
70:      */
71:     private java.lang.String _xmlName;
72:     /**
73:      * Field _identity.
74:      */
75:     private org.exolab.castor.xml.XMLFieldDescriptor _identity;
76:
77:
78:     //-----/
79:     //- Constructors -/
80:     //-----/
81:     public SiteHeaderDescriptor() {
82:         super();
83:         _xmlName = "SiteHeader";
84:         _elementDefinition = true;
85:
86:         //-- set grouping compositor
87:         setCompositorAsSequence();
88:         org.exolab.castor.xml.util.XMLFieldDescriptorImpl desc = null;
89:         org.exolab.castor.mapping.FieldHandler handler = null;
90:         org.exolab.castor.xml.FieldValidator fieldValidator = null;
```

BIRODatabaseManager/src/export/descriptors/SiteHeaderDescriptor.java

```
91:         //-- initialize attribute descriptors
92:
93:         //-- initialize element descriptors
94:
95:         //-- _dateHeaderInformationChecked
96:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(org.exolab.castor.types.Date.class,
"_dateHeaderInformationChecked", "DateHeaderInformationChecked", org.exolab.castor.xml.NodeType.Element);
97:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
98:
99:             public java.lang.Object getValue(java.lang.Object object)
100:                 throws IllegalStateException {
101:                 SiteHeader target = (SiteHeader) object;
102:                 return target.getDateHeaderInformationChecked();
103:             }
104:
105:             public void setValue(java.lang.Object object, java.lang.Object value)
106:                 throws IllegalStateException, IllegalArgumentException {
107:                 try {
108:                     SiteHeader target = (SiteHeader) object;
109:                     target.setDateHeaderInformationChecked((org.exolab.castor.types.Date) value);
110:                 } catch (java.lang.Exception ex) {
111:                     throw new IllegalStateException(ex.toString());
112:                 }
113:             }
114:
115:             public java.lang.Object newInstance(java.lang.Object parent) {
116:                 return new org.exolab.castor.types.Date();
117:             }
118:         };
119:         desc.setHandler(handler);
120:         desc.setRequired(true);
121:         desc.setMultivalued(false);
122:         addFieldDescriptor(desc);
123:
124:         //-- validation code for: _dateHeaderInformationChecked
125:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
126:         fieldValidator.setMinOccurs(1);
127:         { //-- local scope
128:             org.exolab.castor.xml.validators.DateTimeValidator typeValidator;
129:             typeValidator = new org.exolab.castor.xml.validators.DateTimeValidator();
130:             fieldValidator.setValidator(typeValidator);
131:         }
132:         desc.setValidator(fieldValidator);
133:         //-- _DS_ID
134:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(DataSource.class, "_DS_ID", "DS_ID",
```

```
org.exolab.castor.xml.NodeType.Element);
135:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
136:
137:             public java.lang.Object getValue(java.lang.Object object)
138:                 throws IllegalStateException {
139:                 SiteHeader target = (SiteHeader) object;
140:                 return target.getDS_ID();
141:             }
142:
143:             public void setValue(java.lang.Object object, java.lang.Object value)
144:                 throws IllegalStateException, IllegalArgumentException {
145:                 try {
146:                     SiteHeader target = (SiteHeader) object;
147:                     target.setDS_ID((DataSource) value);
148:                 } catch (java.lang.Exception ex) {
149:                     throw new IllegalStateException(ex.toString());
150:                 }
151:             }
152:
153:             public java.lang.Object newInstance(java.lang.Object parent) {
154:                 return null;
155:             }
156:         };
157:         handler = new org.exolab.castor.xml.handlers.EnumFieldHandler(DataSource.class, handler);
158:         desc.setImmutable(true);
159:         desc.setHandler(handler);
160:         desc.setRequired(true);
161:         desc.setMultivalued(false);
162:         addFieldDescriptor(desc);
163:
164:         //-- validation code for: _DS_ID
165:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
166:         fieldValidator.setMinOccurs(1);
167:         { //-- local scope
168:         }
169:         desc.setValidator(fieldValidator);
170:         //-- _DS_WEBSITE
171:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.String.class, "_DS_WEBSITE",
"DS_WEBSITE", org.exolab.castor.xml.NodeType.Element);
172:         desc.setImmutable(true);
173:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
174:
175:             public java.lang.Object getValue(java.lang.Object object)
176:                 throws IllegalStateException {
177:                 SiteHeader target = (SiteHeader) object;
```

BIRODatabaseManager/src/export/descriptors/SiteHeaderDescriptor.java

```
178:         return target.getDS_WEBSITE();
179:     }
180:
181:     public void setValue(java.lang.Object object, java.lang.Object value)
182:         throws IllegalStateException, IllegalArgumentException {
183:         try {
184:             SiteHeader target = (SiteHeader) object;
185:             target.setDS_WEBSITE((java.lang.String) value);
186:         } catch (java.lang.Exception ex) {
187:             throw new IllegalStateException(ex.toString());
188:         }
189:     }
190:
191:     public java.lang.Object newInstance(java.lang.Object parent) {
192:         return null;
193:     }
194: };
195: desc.setHandler(handler);
196: desc.setMultivalued(false);
197: addFieldDescriptor(desc);
198:
199: //-- validation code for: _DS_WEBSITE
200: fieldValidator = new org.exolab.castor.xml.FieldValidator();
201: { //-- local scope
202:     org.exolab.castor.xml.validators.StringValidator typeValidator;
203:     typeValidator = new org.exolab.castor.xml.validators.StringValidator();
204:     fieldValidator.setValidator(typeValidator);
205:     typeValidator.setWhiteSpace("preserve");
206: }
207: desc.setValidator(fieldValidator);
208:
209: //-- _DS_NAME
210: desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.String.class, "_DS_NAME", "DS_NAME"
, org.exolab.castor.xml.NodeType.Element);
211: desc.setImmutable(true);
212: handler = new org.exolab.castor.xml.XMLFieldHandler() {
213:
214:     public java.lang.Object getValue(java.lang.Object object)
215:         throws IllegalStateException {
216:         SiteHeader target = (SiteHeader) object;
217:         return target.getDS_NAME();
218:     }
219:
220:     public void setValue(java.lang.Object object, java.lang.Object value)
221:         throws IllegalStateException, IllegalArgumentException {
```

```
222:         try {
223:             SiteHeader target = (SiteHeader) object;
224:             target.setDS_NAME((java.lang.String) value);
225:         } catch (java.lang.Exception ex) {
226:             throw new IllegalStateException(ex.toString());
227:         }
228:     }
229:
230:     public java.lang.Object newInstance(java.lang.Object parent) {
231:         return null;
232:     }
233: };
234: desc.setHandler(handler);
235: desc.setRequired(true);
236: desc.setMultivalued(false);
237: addFieldDescriptor(desc);
238: //-- validation code for: _DS_NAME
239: fieldValidator = new org.exolab.castor.xml.FieldValidator();
240: { //-- local scope
241:     org.exolab.castor.xml.validators.StringValidator typeValidator;
242:     typeValidator = new org.exolab.castor.xml.validators.StringValidator();
243:     fieldValidator.setValidator(typeValidator);
244:     typeValidator.setWhiteSpace("preserve");
245: }
246:
247:
248: //-- _DS_ADDRESS_1
249: desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.String.class, "_DS_ADDRESS_1",
"DS_ADDRESS_1", org.exolab.castor.xml.NodeType.Element);
250: desc.setImmutable(true);
251: handler = new org.exolab.castor.xml.XMLFieldHandler() {
252:
253:     public java.lang.Object getValue(java.lang.Object object)
254:         throws IllegalStateException {
255:         SiteHeader target = (SiteHeader) object;
256:         return target.getDS_ADDRESS_1();
257:     }
258:
259:     public void setValue(java.lang.Object object, java.lang.Object value)
260:         throws IllegalStateException, IllegalArgumentException {
261:         try {
262:             SiteHeader target = (SiteHeader) object;
263:             target.setDS_ADDRESS_1((java.lang.String) value);
264:         } catch (java.lang.Exception ex) {
265:             throw new IllegalStateException(ex.toString());
```

```
266:         }
267:     }
268:
269:     public java.lang.Object newInstance(java.lang.Object parent) {
270:         return null;
271:     }
272: };
273: desc.setHandler(handler);
274: desc.setRequired(true);
275: desc.setMultivalued(false);
276: addFieldDescriptor(desc);
277:
278: //-- validation code for: _DS_ADDRESS_1
279: fieldValidator = new org.exolab.castor.xml.FieldValidator();
280: fieldValidator.setMinOccurs(1);
281: { //-- local scope
282:     org.exolab.castor.xml.validators.StringValidator typeValidator;
283:     typeValidator = new org.exolab.castor.xml.validators.StringValidator();
284:     fieldValidator.setValidator(typeValidator);
285:     typeValidator.setWhiteSpace("preserve");
286: }
287: desc.setValidator(fieldValidator);
288: //-- _DS_ADDRESS_2
289: desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.String.class, "_DS_ADDRESS_2",
"DS_ADDRESS_2", org.exolab.castor.xml.NodeType.Element);
290: desc.setImmutable(true);
291: handler = new org.exolab.castor.xml.XMLFieldHandler() {
292:
293:     public java.lang.Object getValue(java.lang.Object object)
294:         throws IllegalStateException {
295:         SiteHeader target = (SiteHeader) object;
296:         return target.getDS_ADDRESS_2();
297:     }
298:
299:     public void setValue(java.lang.Object object, java.lang.Object value)
300:         throws IllegalStateException, IllegalArgumentException {
301:         try {
302:             SiteHeader target = (SiteHeader) object;
303:             target.setDS_ADDRESS_2((java.lang.String) value);
304:         } catch (java.lang.Exception ex) {
305:             throw new IllegalStateException(ex.toString());
306:         }
307:     }
308:
309:     public java.lang.Object newInstance(java.lang.Object parent) {
```

```
310:         return null;
311:     }
312: };
313: desc.setHandler(handler);
314: desc.setRequired(true);
315: desc.setMultivalued(false);
316: addFieldDescriptor(desc);
317:
318:     //-- validation code for: _DS_ADDRESS_2
319:     fieldValidator = new org.exolab.castor.xml.FieldValidator();
320:     fieldValidator.setMinOccurs(1);
321:     { //-- local scope
322:         org.exolab.castor.xml.validators.StringValidator typeValidator;
323:         typeValidator = new org.exolab.castor.xml.validators.StringValidator();
324:         fieldValidator.setValidator(typeValidator);
325:         typeValidator.setWhiteSpace("preserve");
326:     }
327:     desc.setValidator(fieldValidator);
328:     //-- _DS_ADDRESS_3
329:     desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.String.class, "_DS_ADDRESS_3",
"DS_ADDRESS_3", org.exolab.castor.xml.NodeType.Element);
330:     desc.setImmutable(true);
331:     handler = new org.exolab.castor.xml.XMLFieldHandler() {
332:
333:         public java.lang.Object getValue(java.lang.Object object)
334:             throws IllegalStateException {
335:             SiteHeader target = (SiteHeader) object;
336:             return target.getDS_ADDRESS_3();
337:         }
338:
339:         public void setValue(java.lang.Object object, java.lang.Object value)
340:             throws IllegalStateException, IllegalArgumentException {
341:             try {
342:                 SiteHeader target = (SiteHeader) object;
343:                 target.setDS_ADDRESS_3((java.lang.String) value);
344:             } catch (java.lang.Exception ex) {
345:                 throw new IllegalStateException(ex.toString());
346:             }
347:         }
348:
349:         public java.lang.Object newInstance(java.lang.Object parent) {
350:             return null;
351:         }
352:     };
353:     desc.setHandler(handler);
```



```
354:         desc.setMultivalued(false);
355:         addFieldDescriptor(desc);
356:
357:         //-- validation code for: _DS_ADDRESS_3
358:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
359:         { //-- local scope
360:             org.exolab.castor.xml.validators.StringValidator typeValidator;
361:             typeValidator = new org.exolab.castor.xml.validators.StringValidator();
362:             fieldValidator.setValidator(typeValidator);
363:             typeValidator.setWhiteSpace("preserve");
364:         }
365:         desc.setValidator(fieldValidator);
366:         //-- _DS_ADDRESS_4
367:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.String.class, "_DS_ADDRESS_4",
"DS_ADDRESS_4", org.exolab.castor.xml.NodeType.Element);
368:         desc.setImmutable(true);
369:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
370:
371:             public java.lang.Object getValue(java.lang.Object object)
372:                 throws IllegalStateException {
373:                 SiteHeader target = (SiteHeader) object;
374:                 return target.getDS_ADDRESS_4();
375:             }
376:
377:             public void setValue(java.lang.Object object, java.lang.Object value)
378:                 throws IllegalStateException, IllegalArgumentException {
379:                 try {
380:                     SiteHeader target = (SiteHeader) object;
381:                     target.setDS_ADDRESS_4((java.lang.String) value);
382:                 } catch (java.lang.Exception ex) {
383:                     throw new IllegalStateException(ex.toString());
384:                 }
385:             }
386:
387:             public java.lang.Object newInstance(java.lang.Object parent) {
388:                 return null;
389:             }
390:         };
391:         desc.setHandler(handler);
392:         desc.setMultivalued(false);
393:         addFieldDescriptor(desc);
394:
395:         //-- validation code for: _DS_ADDRESS_4
396:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
397:         { //-- local scope
```

```
398:         org.exolab.castor.xml.validators.StringValidator typeValidator;
399:         typeValidator = new org.exolab.castor.xml.validators.StringValidator();
400:         fieldValidator.setValidator(typeValidator);
401:         typeValidator.setWhiteSpace("preserve");
402:     }
403:     desc.setValidator(fieldValidator);
404:     //-- _DS_POST_CODE
405:     desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.String.class, "_DS_POST_CODE",
"DS_POST_CODE", org.exolab.castor.xml.NodeType.Element);
406:     desc.setImmutable(true);
407:     handler = new org.exolab.castor.xml.XMLFieldHandler() {
408:
409:         public java.lang.Object getValue(java.lang.Object object)
410:             throws IllegalStateException {
411:             SiteHeader target = (SiteHeader) object;
412:             return target.getDS_POST_CODE();
413:         }
414:
415:         public void setValue(java.lang.Object object, java.lang.Object value)
416:             throws IllegalStateException, IllegalArgumentException {
417:             try {
418:                 SiteHeader target = (SiteHeader) object;
419:                 target.setDS_POST_CODE((java.lang.String) value);
420:             } catch (java.lang.Exception ex) {
421:                 throw new IllegalStateException(ex.toString());
422:             }
423:         }
424:
425:         public java.lang.Object newInstance(java.lang.Object parent) {
426:             return null;
427:         }
428:     };
429:     desc.setHandler(handler);
430:     desc.setMultivalued(false);
431:     addFieldDescriptor(desc);
432:
433:     //-- validation code for: _DS_POST_CODE
434:     fieldValidator = new org.exolab.castor.xml.FieldValidator();
435:     { //-- local scope
436:         org.exolab.castor.xml.validators.StringValidator typeValidator;
437:         typeValidator = new org.exolab.castor.xml.validators.StringValidator();
438:         fieldValidator.setValidator(typeValidator);
439:         typeValidator.setWhiteSpace("preserve");
440:     }
441:     desc.setValidator(fieldValidator);
```

```
442:         //-- _DS_COUNTRY
443:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.String.class, "_DS_COUNTRY",
"DS_COUNTRY", org.exolab.castor.xml.NodeType.Element);
444:         desc.setImmutable(true);
445:         handler = new org.exolab.castor.xml.XMLFieldHandler() {
446:
447:             public java.lang.Object getValue(java.lang.Object object)
448:                 throws IllegalStateException {
449:                 SiteHeader target = (SiteHeader) object;
450:                 return target.getDS_COUNTRY();
451:             }
452:
453:             public void setValue(java.lang.Object object, java.lang.Object value)
454:                 throws IllegalStateException, IllegalArgumentException {
455:                 try {
456:                     SiteHeader target = (SiteHeader) object;
457:                     target.setDS_COUNTRY((java.lang.String) value);
458:                 } catch (java.lang.Exception ex) {
459:                     throw new IllegalStateException(ex.toString());
460:                 }
461:             }
462:
463:             public java.lang.Object newInstance(java.lang.Object parent) {
464:                 return null;
465:             }
466:         };
467:         desc.setHandler(handler);
468:         desc.setRequired(true);
469:         desc.setMultivalued(false);
470:         addFieldDescriptor(desc);
471:
472:         //-- validation code for: _DS_COUNTRY
473:         fieldValidator = new org.exolab.castor.xml.FieldValidator();
474:         fieldValidator.setMinOccurs(1);
475:         { //-- local scope
476:             org.exolab.castor.xml.validators.StringValidator typeValidator;
477:             typeValidator = new org.exolab.castor.xml.validators.StringValidator();
478:             fieldValidator.setValidator(typeValidator);
479:             typeValidator.setWhiteSpace("preserve");
480:         }
481:         desc.setValidator(fieldValidator);
482:         //-- _DS_C_CONTACT
483:         desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.String.class, "_DS_C_CONTACT",
"DS_C_CONTACT", org.exolab.castor.xml.NodeType.Element);
484:         desc.setImmutable(true);
```

```
485:     handler = new org.exolab.castor.xml.XMLFieldHandler() {
486:
487:         public java.lang.Object getValue(java.lang.Object object)
488:             throws IllegalStateException {
489:             SiteHeader target = (SiteHeader) object;
490:             return target.getDS_C_CONTACT();
491:         }
492:
493:         public void setValue(java.lang.Object object, java.lang.Object value)
494:             throws IllegalStateException, IllegalArgumentException {
495:             try {
496:                 SiteHeader target = (SiteHeader) object;
497:                 target.setDS_C_CONTACT((java.lang.String) value);
498:             } catch (java.lang.Exception ex) {
499:                 throw new IllegalStateException(ex.toString());
500:             }
501:         }
502:
503:         public java.lang.Object newInstance(java.lang.Object parent) {
504:             return null;
505:         }
506:     };
507: desc.setHandler(handler);
508: desc.setRequired(true);
509: desc.setMultivalued(false);
510: addFieldDescriptor(desc);
511:
512:     //-- validation code for: _DS_C_CONTACT
513:     fieldValidator = new org.exolab.castor.xml.FieldValidator();
514:     fieldValidator.setMinOccurs(1);
515:     { //-- local scope
516:         org.exolab.castor.xml.validators.StringValidator typeValidator;
517:         typeValidator = new org.exolab.castor.xml.validators.StringValidator();
518:         fieldValidator.setValidator(typeValidator);
519:         typeValidator.setWhiteSpace("preserve");
520:     }
521:     desc.setValidator(fieldValidator);
522:     //-- _DS_C_EMAIL
523:     desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.String.class, "_DS_C_EMAIL",
"DS_C_EMAIL", org.exolab.castor.xml.NodeType.Element);
524:     desc.setImmutable(true);
525:     handler = new org.exolab.castor.xml.XMLFieldHandler() {
526:
527:         public java.lang.Object getValue(java.lang.Object object)
528:             throws IllegalStateException {
```

BIRODatabaseManager/src/export/descriptors/SiteHeaderDescriptor.java

```
529:         SiteHeader target = (SiteHeader) object;
530:         return target.getDS_C_EMAIL();
531:     }
532:
533:     public void setValue(java.lang.Object object, java.lang.Object value)
534:         throws IllegalStateException, IllegalArgumentException {
535:         try {
536:             SiteHeader target = (SiteHeader) object;
537:             target.setDS_C_EMAIL((java.lang.String) value);
538:         } catch (java.lang.Exception ex) {
539:             throw new IllegalStateException(ex.toString());
540:         }
541:     }
542:
543:     public java.lang.Object newInstance(java.lang.Object parent) {
544:         return null;
545:     }
546: };
547: desc.setHandler(handler);
548: desc.setRequired(true);
549: desc.setMultivalued(false);
550: addFieldDescriptor(desc);
551:
552: //-- validation code for: _DS_C_EMAIL
553: fieldValidator = new org.exolab.castor.xml.FieldValidator();
554: fieldValidator.setMinOccurs(1);
555: { //-- local scope
556:     org.exolab.castor.xml.validators.StringValidator typeValidator;
557:     typeValidator = new org.exolab.castor.xml.validators.StringValidator();
558:     fieldValidator.setValidator(typeValidator);
559:     typeValidator.setWhiteSpace("preserve");
560: }
561: desc.setValidator(fieldValidator);
562: //-- _DS_T_CONTACT
563: desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.String.class, "_DS_T_CONTACT",
"DS_T_CONTACT", org.exolab.castor.xml.NodeType.Element);
564: desc.setImmutable(true);
565: handler = new org.exolab.castor.xml.XMLFieldHandler() {
566:
567:     public java.lang.Object getValue(java.lang.Object object)
568:         throws IllegalStateException {
569:         SiteHeader target = (SiteHeader) object;
570:         return target.getDS_T_CONTACT();
571:     }
572:
```

BIRODatabaseManager/src/export/descriptors/SiteHeaderDescriptor.java

```
573:         public void setValue(java.lang.Object object, java.lang.Object value)
574:             throws IllegalStateException, IllegalArgumentException {
575:             try {
576:                 SiteHeader target = (SiteHeader) object;
577:                 target.setDS_T_CONTACT((java.lang.String) value);
578:             } catch (java.lang.Exception ex) {
579:                 throw new IllegalStateException(ex.toString());
580:             }
581:         }
582:
583:         public java.lang.Object newInstance(java.lang.Object parent) {
584:             return null;
585:         }
586:     };
587:     desc.setHandler(handler);
588:     desc.setRequired(true);
589:     desc.setMultivalued(false);
590:     addFieldDescriptor(desc);
591:
592:     //-- validation code for: _DS_T_CONTACT
593:     fieldValidator = new org.exolab.castor.xml.FieldValidator();
594:     fieldValidator.setMinOccurs(1);
595:     { //-- local scope
596:         org.exolab.castor.xml.validators.StringValidator typeValidator;
597:         typeValidator = new org.exolab.castor.xml.validators.StringValidator();
598:         fieldValidator.setValidator(typeValidator);
599:         typeValidator.setWhiteSpace("preserve");
600:     }
601:     desc.setValidator(fieldValidator);
602:     //-- _DS_T_EMAIL
603:     desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.String.class, "_DS_T_EMAIL",
"DS_T_EMAIL", org.exolab.castor.xml.NodeType.Element);
604:     desc.setImmutable(true);
605:     handler = new org.exolab.castor.xml.XMLFieldHandler() {
606:
607:         public java.lang.Object getValue(java.lang.Object object)
608:             throws IllegalStateException {
609:             SiteHeader target = (SiteHeader) object;
610:             return target.getDS_T_EMAIL();
611:         }
612:
613:         public void setValue(java.lang.Object object, java.lang.Object value)
614:             throws IllegalStateException, IllegalArgumentException {
615:             try {
616:                 SiteHeader target = (SiteHeader) object;
```

```
617:         target.setDS_T_EMAIL((java.lang.String) value);
618:     } catch (java.lang.Exception ex) {
619:         throw new IllegalStateException(ex.toString());
620:     }
621: }
622:
623: public java.lang.Object newInstance(java.lang.Object parent) {
624:     return null;
625: }
626: };
627: desc.setHandler(handler);
628: desc.setRequired(true);
629: desc.setMultivalued(false);
630: addFieldDescriptor(desc);
631:
632: //-- validation code for: _DS_T_EMAIL
633: fieldValidator = new org.exolab.castor.xml.FieldValidator();
634: fieldValidator.setMinOccurs(1);
635: { //-- local scope
636:     org.exolab.castor.xml.validators.StringValidator typeValidator;
637:     typeValidator = new org.exolab.castor.xml.validators.StringValidator();
638:     fieldValidator.setValidator(typeValidator);
639:     typeValidator.setWhiteSpace("preserve");
640: }
641: desc.setValidator(fieldValidator);
642: //-- _headerComments
643: desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.String.class, "_headerComments",
"HeaderComments", org.exolab.castor.xml.NodeType.Element);
644: desc.setImmutable(true);
645: handler = new org.exolab.castor.xml.XMLFieldHandler() {
646:
647:     public java.lang.Object getValue(java.lang.Object object)
648:         throws IllegalStateException {
649:         SiteHeader target = (SiteHeader) object;
650:         return target.getHeaderComments();
651:     }
652:
653:     public void setValue(java.lang.Object object, java.lang.Object value)
654:         throws IllegalStateException, IllegalArgumentException {
655:         try {
656:             SiteHeader target = (SiteHeader) object;
657:             target.setHeaderComments((java.lang.String) value);
658:         } catch (java.lang.Exception ex) {
659:             throw new IllegalStateException(ex.toString());
660:         }
661:     }
662: }
```

```
661:         }
662:
663:         public java.lang.Object newInstance(java.lang.Object parent) {
664:             return null;
665:         }
666:     };
667:     desc.setHandler(handler);
668:     desc.setMultivalued(false);
669:     addFieldDescriptor(desc);
670:
671:     //-- validation code for: _headerComments
672:     fieldValidator = new org.exolab.castor.xml.FieldValidator();
673:     { //-- local scope
674:         org.exolab.castor.xml.validators.StringValidator typeValidator;
675:         typeValidator = new org.exolab.castor.xml.validators.StringValidator();
676:         fieldValidator.setValidator(typeValidator);
677:         typeValidator.setWhiteSpace("preserve");
678:     }
679:     desc.setValidator(fieldValidator);
680: }
681:
682:
683: //-----/
684: //- Methods -/
685: //-----/
686: /**
687:  * Method getAccessMode.
688:  *
689:  * @return the access mode specified for this class.
690:  */
691: public org.exolab.castor.mapping.AccessMode getAccessMode() {
692:     return null;
693: }
694:
695: /**
696:  * Method getIdentity.
697:  *
698:  * @return the identity field, null if this class has no
699:  * identity.
700:  */
701: public org.exolab.castor.mapping.FieldDescriptor getIdentity() {
702:     return _identity;
703: }
704:
705: /**
```



```
706:      * Method getJavaClass.
707:      *
708:      * @return the Java class represented by this descriptor.
709:      */
710: public java.lang.Class getJavaClass() {
711:     return export.SiteHeader.class;
712: }
713:
714: /**
715:  * Method getNameSpacePrefix.
716:  *
717:  * @return the namespace prefix to use when marshaling as XML.
718:  */
719: public java.lang.String getNameSpacePrefix() {
720:     return _nsPrefix;
721: }
722:
723: /**
724:  * Method getNameSpaceURI.
725:  *
726:  * @return the namespace URI used when marshaling and
727:  * unmarshaling as XML.
728:  */
729: public java.lang.String getNameSpaceURI() {
730:     return _nsURI;
731: }
732:
733: /**
734:  * Method getValidator.
735:  *
736:  * @return a specific validator for the class described by this
737:  * ClassDescriptor.
738:  */
739: public org.exolab.castor.xml.TypeValidator getValidator() {
740:     return this;
741: }
742:
743: /**
744:  * Method getXMLName.
745:  *
746:  * @return the XML Name for the Class being described.
747:  */
748: public java.lang.String getXMLName() {
749:     return _xmlName;
750: }
```

```
751:
752:  /**
753:   * Method isElementDefinition.
754:   *
755:   * @return true if XML schema definition of this Class is that
756:   * of a global
757:   * element or element with anonymous type definition.
758:   */
759:  public boolean isElementDefinition() {
760:      return _elementDefinition;
761:  }
762: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      SiteProfileDescriptor.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10: * the Free Software Foundation; either version 2, or (at your option)
11: * any later version.
12: *
13: * This file is distributed in the hope that it will be useful,
14: * but WITHOUT ANY WARRANTY; without even the implied warranty of
15: * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16: * GNU General Public License for more details.
17: *
18: * You should have received a copy of the GNU General Public License
19: * along with this file; see the file COPYING. If not, write to
20: * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21: *
22: * In short: you may use this file any way you like, as long as you
23: * don't charge money for it, remove this notice, or hold anyone liable
24: * for its results.
25: *
26:
27: * GPL Copyright, The BIRO Project
28: *
29: **/
30:
31: /**
32:  * This class was automatically generated with
33:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
34:  * Schema.
35:  * $Id$
36:  */
37:
38: package export.descriptors;
39:
40:  //-----/
41:  //- Imported classes and packages -/
42:  //-----/
43:
44: import export.SiteProfile;
45:
```

```
46: /**
47:  * Class SiteProfileDescriptor.
48:  *
49:  * @version $Revision$ $Date$
50:  */
51: public class SiteProfileDescriptor extends org.exolab.castor.xml.util.XMLClassDescriptorImpl {
52:
53:
54:     //-----/
55:     //- Class/Member Variables -/
56:     //-----/
57:
58:     /**
59:      * Field _elementDefinition.
60:      */
61:     private boolean _elementDefinition;
62:
63:     /**
64:      * Field _nsPrefix.
65:      */
66:     private java.lang.String _nsPrefix;
67:
68:     /**
69:      * Field _nsURI.
70:      */
71:     private java.lang.String _nsURI;
72:
73:     /**
74:      * Field _xmlName.
75:      */
76:     private java.lang.String _xmlName;
77:
78:     /**
79:      * Field _identity.
80:      */
81:     private org.exolab.castor.xml.XMLFieldDescriptor _identity;
82:
83:
84:     //-----/
85:     //- Constructors -/
86:     //-----/
87:
88:     public SiteProfileDescriptor() {
89:         super();
90:         _xmlName = "SiteProfile";
```

```
91:         _elementDefinition = true;
92:
93:         //-- set grouping compositor
94:         setCompositorAsSequence();
95:         org.exolab.castor.xml.util.XMLFieldDescriptorImpl desc          = null;
96:         org.exolab.castor.mapping.FieldHandler handler              = null;
97:         org.exolab.castor.xml.FieldValidator fieldValidator = null;
98:         //-- initialize attribute descriptors
99:
100:        //-- initialize element descriptors
101:
102:        //-- _dateProfileInformationChecked
103:        desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(org.exolab.castor.types.Date.class,
"_dateProfileInformationChecked", "DateProfileInformationChecked", org.exolab.castor.xml.NodeType.Element);
104:        handler = new org.exolab.castor.xml.XMLFieldHandler() {
105:            public java.lang.Object getValue( java.lang.Object object )
106:                throws IllegalStateException
107:            {
108:                SiteProfile target = (SiteProfile) object;
109:                return target.getDateProfileInformationChecked();
110:            }
111:            @Override
112:            public void setValue( java.lang.Object object, java.lang.Object value)
113:                throws IllegalStateException, IllegalArgumentException
114:            {
115:                try {
116:                    SiteProfile target = (SiteProfile) object;
117:                    target.setDateProfileInformationChecked( (org.exolab.castor.types.Date) value);
118:                } catch (java.lang.Exception ex) {
119:                    throw new IllegalStateException(ex.toString());
120:                }
121:            }
122:            public java.lang.Object newInstance(java.lang.Object parent) {
123:                return new org.exolab.castor.types.Date();
124:            }
125:        };
126:        desc.setHandler(handler);
127:        //desc.setRequired(true);
128:        desc.setMultivalued(false);
129:        addFieldDescriptor(desc);
130:
131:        //-- validation code for: _dateProfileInformationChecked
132:        fieldValidator = new org.exolab.castor.xml.FieldValidator();
133:        fieldValidator.setMinOccurs(1);
134:        { //-- local scope
```

BIRODatabaseManager/src/export/descriptors/SiteProfileDescriptor.java

```
135:         org.exolab.castor.xml.validators.DateTimeValidator typeValidator;
136:         typeValidator = new org.exolab.castor.xml.validators.DateTimeValidator();
137:         fieldValidator.setValidator(typeValidator);
138:     }
139:     desc.setValidator(fieldValidator);
140:     //-- _DS_TYPE
141:     desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(export.types.SiteType.class, "_DS_TYPE",
"DS_TYPE", org.exolab.castor.xml.NodeType.Element);
142:     handler = new org.exolab.castor.xml.XMLFieldHandler() {
143:         public java.lang.Object getValue( java.lang.Object object )
144:             throws IllegalStateException
145:         {
146:             SiteProfile target = (SiteProfile) object;
147:             return target.getDS_TYPE();
148:         }
149:         public void setValue( java.lang.Object object, java.lang.Object value)
150:             throws IllegalStateException, IllegalArgumentException
151:         {
152:             try {
153:                 SiteProfile target = (SiteProfile) object;
154:                 target.setDS_TYPE( (export.types.SiteType) value);
155:             } catch (java.lang.Exception ex) {
156:                 throw new IllegalStateException(ex.toString());
157:             }
158:         }
159:         public java.lang.Object newInstance(java.lang.Object parent) {
160:             return null;
161:         }
162:     };
163:     handler = new org.exolab.castor.xml.handlers.EnumFieldHandler(export.types.SiteType.class, handler);
164:     desc.setImmutable(true);
165:     desc.setHandler(handler);
166:     //desc.setRequired(true);
167:     desc.setMultivalued(false);
168:     addFieldDescriptor(desc);
169:
170:     //-- validation code for: _DS_TYPE
171:     fieldValidator = new org.exolab.castor.xml.FieldValidator();
172:     fieldValidator.setMinOccurs(1);
173:     { //-- local scope
174:     }
175:     desc.setValidator(fieldValidator);
176:     //-- _DS_DENOM
177:     desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.Long.TYPE, "_DS_DENOM", "DS_DENOM",
org.exolab.castor.xml.NodeType.Element);
```

BIRODatabaseManager/src/export/descriptors/SiteProfileDescriptor.java

```
178:     handler = new org.exolab.castor.xml.XMLFieldHandler() {
179:         public java.lang.Object getValue( java.lang.Object object )
180:             throws IllegalStateException
181:         {
182:             SiteProfile target = (SiteProfile) object;
183:             if (!target.hasDS_DENOM()) { return null; }
184:             return new java.lang.Long(target.getDS_DENOM());
185:         }
186:         public void setValue( java.lang.Object object, java.lang.Object value)
187:             throws IllegalStateException, IllegalArgumentException
188:         {
189:             try {
190:                 SiteProfile target = (SiteProfile) object;
191:                 // ignore null values for non optional primitives
192:                 if (value == null) { return; }
193:
194:                 target.setDS_DENOM( ((java.lang.Long) value).longValue());
195:             } catch (java.lang.Exception ex) {
196:                 throw new IllegalStateException(ex.toString());
197:             }
198:         }
199:         public java.lang.Object newInstance(java.lang.Object parent) {
200:             return null;
201:         }
202:     };
203:     desc.setHandler(handler);
204:     //desc.setRequired(true);
205:     desc.setMultivalued(false);
206:     addFieldDescriptor(desc);
207:
208:     //-- validation code for: _DS_DENOM
209:     fieldValidator = new org.exolab.castor.xml.FieldValidator();
210:     fieldValidator.setMinOccurs(1);
211:     { //-- local scope
212:         org.exolab.castor.xml.validators.LongValidator typeValidator;
213:         typeValidator = new org.exolab.castor.xml.validators.LongValidator();
214:         fieldValidator.setValidator(typeValidator);
215:         //typeValidator.setMinInclusive(1L);
216:     }
217:     desc.setValidator(fieldValidator);
218:     //-- _DS_AREA
219:     desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl( java.lang.Long.TYPE, "_DS_AREA", "DS_AREA",
org.exolab.castor.xml.NodeType.Element);
220:     handler = new org.exolab.castor.xml.XMLFieldHandler() {
221:         public java.lang.Object getValue( java.lang.Object object )
```

BIRODatabaseManager/src/export/descriptors/SiteProfileDescriptor.java

```
222:         throws IllegalStateException
223:     {
224:         SiteProfile target = (SiteProfile) object;
225:         if (!target.hasDS_AREA()) { return null; }
226:         return new java.lang.Long(target.getDS_AREA());
227:     }
228:     public void setValue( java.lang.Object object, java.lang.Object value)
229:         throws IllegalStateException, IllegalArgumentException
230:     {
231:         try {
232:             SiteProfile target = (SiteProfile) object;
233:             // ignore null values for non optional primitives
234:             if (value == null) { return; }
235:
236:             target.setDS_AREA( ((java.lang.Long) value).longValue());
237:         } catch (java.lang.Exception ex) {
238:             throw new IllegalStateException(ex.toString());
239:         }
240:     }
241:     public java.lang.Object newInstance(java.lang.Object parent) {
242:         return null;
243:     }
244: };
245: desc.setHandler(handler);
246: //desc.setRequired(true);
247: desc.setMultivalued(false);
248: addFieldDescriptor(desc);
249:
250: //-- validation code for: _DS_AREA
251: fieldValidator = new org.exolab.castor.xml.FieldValidator();
252: fieldValidator.setMinOccurs(1);
253: { //-- local scope
254:     org.exolab.castor.xml.validators.LongValidator typeValidator;
255:     typeValidator = new org.exolab.castor.xml.validators.LongValidator();
256:     fieldValidator.setValidator(typeValidator);
257:     //typeValidator.setMinInclusive(1L);
258: }
259: desc.setValidator(fieldValidator);
260: //-- _DS_BEDS
261: desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.Long.TYPE, "_DS_BEDS", "DS_BEDS",
org.exolab.castor.xml.NodeType.Element);
262:     handler = new org.exolab.castor.xml.XMLFieldHandler() {
263:         public java.lang.Object getValue( java.lang.Object object )
264:             throws IllegalStateException
265:         {
```


BIRODatabaseManager/src/export/descriptors/SiteProfileDescriptor.java

```
266:         SiteProfile target = (SiteProfile) object;
267:         if (!target.hasDS_BEDS()) { return null; }
268:         return new java.lang.Long(target.getDS_BEDS());
269:     }
270:     public void setValue( java.lang.Object object, java.lang.Object value)
271:         throws IllegalStateException, IllegalArgumentException
272:     {
273:         try {
274:             SiteProfile target = (SiteProfile) object;
275:             // ignore null values for non optional primitives
276:             if (value == null) { return; }
277:
278:             target.setDS_BEDS( ((java.lang.Long) value).longValue());
279:         } catch (java.lang.Exception ex) {
280:             throw new IllegalStateException(ex.toString());
281:         }
282:     }
283:     public java.lang.Object newInstance(java.lang.Object parent) {
284:         return null;
285:     }
286: };
287: desc.setHandler(handler);
288: //desc.setRequired(true);
289: desc.setMultivalued(false);
290: addFieldDescriptor(desc);
291:
292: //-- validation code for: _DS_BEDS
293: fieldValidator = new org.exolab.castor.xml.FieldValidator();
294: fieldValidator.setMinOccurs(1);
295: { //-- local scope
296:     org.exolab.castor.xml.validators.LongValidator typeValidator;
297:     typeValidator = new org.exolab.castor.xml.validators.LongValidator();
298:     fieldValidator.setValidator(typeValidator);
299:     //typeValidator.setMinInclusive(1L);
300: }
301: desc.setValidator(fieldValidator);
302: //-- _DS_PHYSICIANS
303: desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl( java.lang.Long.TYPE, "_DS_PHYSICIANS",
"DS_PHYSICIANS", org.exolab.castor.xml.NodeType.Element);
304: handler = new org.exolab.castor.xml.XMLFieldHandler() {
305:     public java.lang.Object getValue( java.lang.Object object )
306:         throws IllegalStateException
307:     {
308:         SiteProfile target = (SiteProfile) object;
309:         if (!target.hasDS_PHYSICIANS()) { return null; }
```

BIRODatabaseManager/src/export/descriptors/SiteProfileDescriptor.java

```
310:         return new java.lang.Long(target.getDS_PHYSICIANS());
311:     }
312:     public void setValue( java.lang.Object object, java.lang.Object value)
313:         throws IllegalStateException, IllegalArgumentException
314:     {
315:         try {
316:             SiteProfile target = (SiteProfile) object;
317:             // ignore null values for non optional primitives
318:             if (value == null) { return; }
319:
320:             target.setDS_PHYSICIANS( ((java.lang.Long) value).longValue());
321:         } catch (java.lang.Exception ex) {
322:             throw new IllegalStateException(ex.toString());
323:         }
324:     }
325:     public java.lang.Object newInstance(java.lang.Object parent) {
326:         return null;
327:     }
328: };
329: desc.setHandler(handler);
330: //desc.setRequired(true);
331: desc.setMultivalued(false);
332: addFieldDescriptor(desc);
333:
334: //-- validation code for: _DS_PHYSICIANS
335: fieldValidator = new org.exolab.castor.xml.FieldValidator();
336: fieldValidator.setMinOccurs(1);
337: { //-- local scope
338:     org.exolab.castor.xml.validators.LongValidator typeValidator;
339:     typeValidator = new org.exolab.castor.xml.validators.LongValidator();
340:     fieldValidator.setValidator(typeValidator);
341:     //typeValidator.setMinInclusive(1L);
342: }
343: desc.setValidator(fieldValidator);
344: //-- _DS_DIABETOLOGISTS
345: desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl( java.lang.Long.TYPE, "_DS_DIABETOLOGISTS",
"DS_DIABETOLOGISTS", org.exolab.castor.xml.NodeType.Element);
346: handler = new org.exolab.castor.xml.XMLFieldHandler() {
347:     public java.lang.Object getValue( java.lang.Object object )
348:         throws IllegalStateException
349:     {
350:         SiteProfile target = (SiteProfile) object;
351:         if (!target.hasDS_DIABETOLOGISTS()) { return null; }
352:         return new java.lang.Long(target.getDS_DIABETOLOGISTS());
353:     }
}
```

BIRODatabaseManager/src/export/descriptors/SiteProfileDescriptor.java

```
354:         public void setValue( java.lang.Object object, java.lang.Object value)
355:             throws IllegalStateException, IllegalArgumentException
356:         {
357:             try {
358:                 SiteProfile target = (SiteProfile) object;
359:                 // ignore null values for non optional primitives
360:                 if (value == null) { return; }
361:
362:                 target.setDS_DIABETOLOGISTS( ((java.lang.Long) value).longValue());
363:             } catch (java.lang.Exception ex) {
364:                 throw new IllegalStateException(ex.toString());
365:             }
366:         }
367:         public java.lang.Object newInstance(java.lang.Object parent) {
368:             return null;
369:         }
370:     };
371:     desc.setHandler(handler);
372:     //desc.setRequired(true);
373:     desc.setMultivalued(false);
374:     addFieldDescriptor(desc);
375:
376:     //-- validation code for: _DS_DIABETOLOGISTS
377:     fieldValidator = new org.exolab.castor.xml.FieldValidator();
378:     fieldValidator.setMinOccurs(1);
379:     { //-- local scope
380:         org.exolab.castor.xml.validators.LongValidator typeValidator;
381:         typeValidator = new org.exolab.castor.xml.validators.LongValidator();
382:         fieldValidator.setValidator(typeValidator);
383:         //typeValidator.setMinInclusive(1L);
384:     }
385:     desc.setValidator(fieldValidator);
386:     //-- _DS_DOCTORS
387:     desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.Long.TYPE, "_DS_DOCTORS",
"DS_DOCTORS", org.exolab.castor.xml.NodeType.Element);
388:     handler = new org.exolab.castor.xml.XMLFieldHandler() {
389:         public java.lang.Object getValue( java.lang.Object object )
390:             throws IllegalStateException
391:         {
392:             SiteProfile target = (SiteProfile) object;
393:             if (!target.hasDS_DOCTORS()) { return null; }
394:             return new java.lang.Long(target.getDS_DOCTORS());
395:         }
396:     }
397:     public void setValue( java.lang.Object object, java.lang.Object value)
        throws IllegalStateException, IllegalArgumentException
```

```
398:         {
399:             try {
400:                 SiteProfile target = (SiteProfile) object;
401:                 // ignore null values for non optional primitives
402:                 if (value == null) { return; }
403:
404:                 target.setDS_DOCTORS( ((java.lang.Long) value).longValue());
405:             } catch (java.lang.Exception ex) {
406:                 throw new IllegalStateException(ex.toString());
407:             }
408:         }
409:         public java.lang.Object newInstance(java.lang.Object parent) {
410:             return null;
411:         }
412:     };
413:     desc.setHandler(handler);
414:     //desc.setRequired(true);
415:     desc.setMultivalued(false);
416:     addFieldDescriptor(desc);
417:
418:     //-- validation code for: _DS_DOCTORS
419:     fieldValidator = new org.exolab.castor.xml.FieldValidator();
420:     fieldValidator.setMinOccurs(1);
421:     { //-- local scope
422:         org.exolab.castor.xml.validators.LongValidator typeValidator;
423:         typeValidator = new org.exolab.castor.xml.validators.LongValidator();
424:         fieldValidator.setValidator(typeValidator);
425:         //typeValidator.setMinInclusive(1L);
426:     }
427:     desc.setValidator(fieldValidator);
428:     //-- _DS_DSN
429:     desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.Long.TYPE, "_DS_DSN", "DS_DSN",
org.exolab.castor.xml.NodeType.Element);
430:     handler = new org.exolab.castor.xml.XMLFieldHandler() {
431:         public java.lang.Object getValue( java.lang.Object object )
432:             throws IllegalStateException
433:         {
434:             SiteProfile target = (SiteProfile) object;
435:             if (!target.hasDS_DSN()) { return null; }
436:             return new java.lang.Long(target.getDS_DSN());
437:         }
438:         public void setValue( java.lang.Object object, java.lang.Object value)
439:             throws IllegalStateException, IllegalArgumentException
440:         {
441:             try {
```

BIRODatabaseManager/src/export/descriptors/SiteProfileDescriptor.java

```
442:         SiteProfile target = (SiteProfile) object;
443:         // ignore null values for non optional primitives
444:         if (value == null) { return; }
445:
446:         target.setDS_DSN( ((java.lang.Long) value).longValue());
447:     } catch (java.lang.Exception ex) {
448:         throw new IllegalStateException(ex.toString());
449:     }
450: }
451: public java.lang.Object newInstance(java.lang.Object parent) {
452:     return null;
453: }
454: };
455: desc.setHandler(handler);
456: //desc.setRequired(true);
457: desc.setMultivalued(false);
458: addFieldDescriptor(desc);
459:
460: //-- validation code for: _DS_DSN
461: fieldValidator = new org.exolab.castor.xml.FieldValidator();
462: fieldValidator.setMinOccurs(1);
463: { //-- local scope
464:     org.exolab.castor.xml.validators.LongValidator typeValidator;
465:     typeValidator = new org.exolab.castor.xml.validators.LongValidator();
466:     fieldValidator.setValidator(typeValidator);
467:     //typeValidator.setMinInclusive(1L);
468: }
469: desc.setValidator(fieldValidator);
470: //-- _DS_DMP_PHYSICIANS
471: desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.Long.TYPE, "_DS_DMP_PHYSICIANS",
"DS_DMP_PHYSICIANS", org.exolab.castor.xml.NodeType.Element);
472: handler = new org.exolab.castor.xml.XMLFieldHandler() {
473:     public java.lang.Object getValue( java.lang.Object object )
474:         throws IllegalStateException
475:     {
476:         SiteProfile target = (SiteProfile) object;
477:         if (!target.hasDS_DMP_PHYSICIANS()) { return null; }
478:         return new java.lang.Long(target.getDS_DMP_PHYSICIANS());
479:     }
480:     public void setValue( java.lang.Object object, java.lang.Object value)
481:         throws IllegalStateException, IllegalArgumentException
482:     {
483:         try {
484:             SiteProfile target = (SiteProfile) object;
485:             // ignore null values for non optional primitives
```

BIRODatabaseManager/src/export/descriptors/SiteProfileDescriptor.java

```
486:         if (value == null) { return; }
487:
488:         target.setDS_DMP_PHYSICIANS( ((java.lang.Long) value).longValue());
489:     } catch (java.lang.Exception ex) {
490:         throw new IllegalStateException(ex.toString());
491:     }
492: }
493: public java.lang.Object newInstance(java.lang.Object parent) {
494:     return null;
495: }
496: };
497: desc.setHandler(handler);
498: //desc.setRequired(true);
499: desc.setMultivalued(false);
500: addFieldDescriptor(desc);
501:
502: //-- validation code for: _DS_DMP_PHYSICIANS
503: fieldValidator = new org.exolab.castor.xml.FieldValidator();
504: fieldValidator.setMinOccurs(1);
505: { //-- local scope
506:     org.exolab.castor.xml.validators.LongValidator typeValidator;
507:     typeValidator = new org.exolab.castor.xml.validators.LongValidator();
508:     fieldValidator.setValidator(typeValidator);
509:     //typeValidator.setMinInclusive(1L);
510: }
511: desc.setValidator(fieldValidator);
512: //-- _profileComments
513: desc = new org.exolab.castor.xml.util.XMLFieldDescriptorImpl(java.lang.String.class, "_profileComments",
"ProfileComments", org.exolab.castor.xml.NodeType.Element);
514: desc.setImmutable(true);
515: handler = new org.exolab.castor.xml.XMLFieldHandler() {
516:     public java.lang.Object getValue( java.lang.Object object )
517:         throws IllegalStateException
518:     {
519:         SiteProfile target = (SiteProfile) object;
520:         return target.getProfileComments();
521:     }
522:     public void setValue( java.lang.Object object, java.lang.Object value)
523:         throws IllegalStateException, IllegalArgumentException
524:     {
525:         try {
526:             SiteProfile target = (SiteProfile) object;
527:             target.setProfileComments( (java.lang.String) value);
528:         } catch (java.lang.Exception ex) {
529:             throw new IllegalStateException(ex.toString());
```

```
530:         }
531:     }
532:     public java.lang.Object newInstance(java.lang.Object parent) {
533:         return null;
534:     }
535: };
536: desc.setHandler(handler);
537: desc.setMultivalued(false);
538: addFieldDescriptor(desc);
539:
540: //-- validation code for: _profileComments
541: fieldValidator = new org.exolab.castor.xml.FieldValidator();
542: { //-- local scope
543:     org.exolab.castor.xml.validators.StringValidator typeValidator;
544:     typeValidator = new org.exolab.castor.xml.validators.StringValidator();
545:     fieldValidator.setValidator(typeValidator);
546:     typeValidator.setWhiteSpace("preserve");
547: }
548: desc.setValidator(fieldValidator);
549: }
550:
551:
552: //-----/
553: //- Methods -/
554: //-----/
555:
556: /**
557:  * Method getAccessMode.
558:  *
559:  * @return the access mode specified for this class.
560:  */
561: public org.exolab.castor.mapping.AccessMode getAccessMode(
562: ) {
563:     return null;
564: }
565:
566: /**
567:  * Method getIdentity.
568:  *
569:  * @return the identity field, null if this class has no
570:  * identity.
571:  */
572: public org.exolab.castor.mapping.FieldDescriptor getIdentity(
573: ) {
574:     return _identity;
```

```
575:     }
576:
577:     /**
578:      * Method getJavaClass.
579:      *
580:      * @return the Java class represented by this descriptor.
581:      */
582:     public java.lang.Class getJavaClass(
583:     ) {
584:         return export.SiteProfile.class;
585:     }
586:
587:     /**
588:      * Method getNameSpacePrefix.
589:      *
590:      * @return the namespace prefix to use when marshaling as XML.
591:      */
592:     public java.lang.String getNameSpacePrefix(
593:     ) {
594:         return _nsPrefix;
595:     }
596:
597:     /**
598:      * Method getNameSpaceURI.
599:      *
600:      * @return the namespace URI used when marshaling and
601:      * unmarshaling as XML.
602:      */
603:     public java.lang.String getNameSpaceURI(
604:     ) {
605:         return _nsURI;
606:     }
607:
608:     /**
609:      * Method getValidator.
610:      *
611:      * @return a specific validator for the class described by this
612:      * ClassDescriptor.
613:      */
614:     public org.exolab.castor.xml.TypeValidator getValidator(
615:     ) {
616:         return this;
617:     }
618:
619:     /**
```



```
620:      * Method getXMLName.
621:      *
622:      * @return the XML Name for the Class being described.
623:      */
624:  public java.lang.String getXMLName(
625:  ) {
626:      return _xmlName;
627:  }
628:
629:  /**
630:   * Method isElementDefinition.
631:   *
632:   * @return true if XML schema definition of this Class is that
633:   * of a global
634:   * element or element with anonymous type definition.
635:   */
636:  public boolean isElementDefinition(
637:  ) {
638:      return _elementDefinition;
639:  }
640:
641: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      ActivityEndReason.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: /*
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36: package export.types;
37:
38: //-----/
39: //- Imported classes and packages -/
40: //-----/
41: import java.util.Hashtable;
42:
43: /**
44:  * Class ActivityEndReason.
45:  *
```

```
46:  * @version $Revision$ $Date$
47:  */
48: public class ActivityEndReason implements java.io.Serializable {
49:
50:
51:     //-----/
52:     //- Class/Member Variables -/
53:     //-----/
54:     /**
55:      * The 1 type
56:      */
57:     public static final int VALUE_1_TYPE = 0;
58:     /**
59:      * The instance of the 1 type
60:      */
61:     public static final ActivityEndReason VALUE_1 = new ActivityEndReason(VALUE_1_TYPE, "1");
62:     /**
63:      * The 2 type
64:      */
65:     public static final int VALUE_2_TYPE = 1;
66:     /**
67:      * The instance of the 2 type
68:      */
69:     public static final ActivityEndReason VALUE_2 = new ActivityEndReason(VALUE_2_TYPE, "2");
70:     /**
71:      * The 3 type
72:      */
73:     public static final int VALUE_3_TYPE = 2;
74:     /**
75:      * The instance of the 3 type
76:      */
77:     public static final ActivityEndReason VALUE_3 = new ActivityEndReason(VALUE_3_TYPE, "3");
78:     /**
79:      * Field _memberTable.
80:      */
81:     private static java.util.Hashtable _memberTable = init();
82:     /**
83:      * Field type.
84:      */
85:     private int type = -1;
86:     /**
87:      * Field stringValue.
88:      */
89:     private java.lang.String stringValue = null;
90:
```

```
91:
92:     //-----/
93:     //- Constructors -/
94:     //-----/
95:
96:     //added by Valentina
97:     public ActivityEndReason() {
98:         super();
99:         this.type = -1;
100:        this.stringValue = null;
101:    }
102:
103:    private ActivityEndReason(final int type, final java.lang.String value) {
104:        super();
105:        this.type = type;
106:        this.stringValue = value;
107:    }
108:
109:
110:    //-----/
111:    //- Methods -/
112:    //-----/
113:    /**
114:     * Method enumerate.Returns an enumeration of all possible
115:     * instances of ActivityEndReason
116:     *
117:     * @return an Enumeration over all possible instances of
118:     * ActivityEndReason
119:     */
120:    public static java.util.Enumeration enumerate() {
121:        return _memberTable.elements();
122:    }
123:
124:    /**
125:     * Method getType.Returns the type of this ActivityEndReason
126:     *
127:     * @return the type of this ActivityEndReason
128:     */
129:    public int getType() {
130:        return this.type;
131:    }
132:
133:    /**
134:     * Method init.
135:     *
```

```
136:      * @return the initialized Hashtable for the member table
137:      */
138:  private static java.util.Hashtable init() {
139:      Hashtable members = new Hashtable();
140:      members.put("1", VALUE_1);
141:      members.put("2", VALUE_2);
142:      members.put("3", VALUE_3);
143:      return members;
144:  }
145:
146:  /**
147:   * Method readResolve. will be called during deserialization to
148:   * replace the deserialized object with the correct constant
149:   * instance.
150:   *
151:   * @return this deserialized object
152:   */
153:  private java.lang.Object readResolve() {
154:      return valueOf(this.stringValue);
155:  }
156:
157:  /**
158:   * Method toString.Returns the String representation of this
159:   * ActivityEndReason
160:   *
161:   * @return the String representation of this ActivityEndReason
162:   */
163:  public java.lang.String toString() {
164:      return this.stringValue;
165:  }
166:
167:  /**
168:   * Method valueOf.Returns a new ActivityEndReason based on the
169:   * given String value.
170:   *
171:   * @param string
172:   * @return the ActivityEndReason value of parameter 'string'
173:   */
174:  public static ActivityEndReason valueOf(final java.lang.String string) {
175:      java.lang.Object obj = null;
176:      if (string != null) {
177:          obj = _memberTable.get(string);
178:      }
179:      if (obj == null) {
180:          String err = "" + string + " is not a valid ActivityEndReason";
```

```
181:         throw new IllegalArgumentException(err);
182:     }
183:     return (ActivityEndReason) obj;
184: }
185:
186: /**
187:  * Method getInstance.
188:  * add by Vale
189:  */
190: public static Object getInstance(String string) {
191:     java.lang.Object obj = null;
192:     if (string != null) {
193:         obj = _memberTable.get(string);
194:     }
195:     if (obj == null) {
196:         String err = "" + string + " is not a valid ActivityEndReason";
197:         throw new IllegalArgumentException(err);
198:     }
199:     return (ActivityEndReason) obj;
200: }
201:
202: /**
203:  * Method setType. Set the type of this ActivityEndReason
204:  * add by Vale
205:  */
206: public void setType(int type) {
207:     this.type = type;
208: }
209:
210: /**
211:  * Method setStringValue. Set the stringValue of this ActivityEndReason
212:  * add by Vale
213:  */
214: public void setStringValue(String stringValue) {
215:     this.stringValue = stringValue;
216: }
217: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      ActivityEndReasonUserType.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: package export.types;
31:
32: import java.io.Serializable;
33: import java.sql.PreparedStatement;
34: import java.sql.ResultSet;
35: import java.sql.SQLException;
36: import java.sql.Types;
37:
38: import org.hibernate.HibernateException;
39: import org.hibernate.usertype.UserType;
40:
41: public class ActivityEndReasonUserType implements UserType {
42:
43:     private static final int[] SQL_TYPES = {Types.VARCHAR};
44:
45:     public int[] sqlTypes() {
```

```
46:         return SQL_TYPES;
47:     }
48:
49:     public Class returnedClass() {
50:         return ActivityEndReason.class;
51:     }
52:
53:     public boolean equals(Object x, Object y) {
54:         return x == y;
55:     }
56:
57:     public Object deepCopy(Object value) {
58:         return value;
59:     }
60:
61:     public boolean isMutable() {
62:         return false;
63:     }
64:
65:     public Object nullSafeGet(ResultSet resultSet, String[] names, Object owner) throws HibernateException,
SQLException {
66:         String name = resultSet.getString(names[0]);
67:         return resultSet.isNull() ? null : ActivityEndReason.getInstance(name);
68:     }
69:
70:     public void nullSafeSet(PreparedStatement statement, Object value, int index) throws HibernateException,
SQLException {
71:
72:         if (value == null) {
73:             statement.setNull(index, Types.VARCHAR);
74:         } else {
75:             statement.setString(index, value.toString());
76:         }
77:     }
78:
79:     public Object assemble(Serializable arg0, Object arg1) throws HibernateException {
80:         // TODO Auto-generated method stub
81:         return null;
82:     }
83:
84:     public Serializable disassemble(Object arg0) throws HibernateException {
85:         // TODO Auto-generated method stub
86:         return null;
87:     }
88:
```



```
89:    public int hashCode(Object arg0) throws HibernateException {
90:        // TODO Auto-generated method stub
91:        return 0;
92:    }
93:
94:    public Object replace(Object arg0, Object arg1, Object arg2) throws HibernateException {
95:        // TODO Auto-generated method stub
96:        return null;
97:    }
98: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      ActivityStartReason.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10: * the Free Software Foundation; either version 2, or (at your option)
11: * any later version.
12: *
13: * This file is distributed in the hope that it will be useful,
14: * but WITHOUT ANY WARRANTY; without even the implied warranty of
15: * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16: * GNU General Public License for more details.
17: *
18: * You should have received a copy of the GNU General Public License
19: * along with this file; see the file COPYING. If not, write to
20: * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21: *
22: * In short: you may use this file any way you like, as long as you
23: * don't charge money for it, remove this notice, or hold anyone liable
24: * for its results.
25: *
26:
27: * GPL Copyright, The BIRO Project
28: *
29: **/
30: /*
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36: package export.types;
37:
38: //-----/
39: //- Imported classes and packages -/
40: //-----/
41: import java.util.Hashtable;
42:
43: /**
44:  * Class ActivityStartReason.
45:  *
```

```
46:  * @version $Revision$ $Date$
47:  */
48: public class ActivityStartReason implements java.io.Serializable {
49:
50:
51:     //-----/
52:     //- Class/Member Variables -/
53:     //-----/
54:     /**
55:      * The 1 type
56:      */
57:     public static final int VALUE_1_TYPE = 0;
58:     /**
59:      * The instance of the 1 type
60:      */
61:     public static final ActivityStartReason VALUE_1 = new ActivityStartReason(VALUE_1_TYPE, "1");
62:     /**
63:      * The 2 type
64:      */
65:     public static final int VALUE_2_TYPE = 1;
66:     /**
67:      * The instance of the 2 type
68:      */
69:     public static final ActivityStartReason VALUE_2 = new ActivityStartReason(VALUE_2_TYPE, "2");
70:     /**
71:      * The 3 type
72:      */
73:     public static final int VALUE_3_TYPE = 2;
74:     /**
75:      * The instance of the 3 type
76:      */
77:     public static final ActivityStartReason VALUE_3 = new ActivityStartReason(VALUE_3_TYPE, "3");
78:     /**
79:      * Field _memberTable.
80:      */
81:     private static java.util.Hashtable _memberTable = init();
82:     /**
83:      * Field type.
84:      */
85:     private int type = -1;
86:     /**
87:      * Field stringValue.
88:      */
89:     private java.lang.String stringValue = null;
90:
```

```
91:
92: //-----/
93: //- Constructors -/
94: //-----/
95: //added by Valentina
96: public ActivityStartReason() {
97:     super();
98:     this.type = -1;
99:     this.stringValue = null;
100: }
101:
102: private ActivityStartReason(final int type, final java.lang.String value) {
103:     super();
104:     this.type = type;
105:     this.stringValue = value;
106: }
107:
108:
109: //-----/
110: //- Methods -/
111: //-----/
112: /**
113:  * Method enumerate.Returns an enumeration of all possible
114:  * instances of ActivityStartReason
115:  *
116:  * @return an Enumeration over all possible instances of
117:  * ActivityStartReason
118:  */
119: public static java.util Enumeration enumerate() {
120:     return _memberTable.elements();
121: }
122:
123: /**
124:  * Method getType.Returns the type of this ActivityStartReason
125:  *
126:  * @return the type of this ActivityStartReason
127:  */
128: public int getType() {
129:     return this.type;
130: }
131:
132: /**
133:  * Method init.
134:  *
135:  * @return the initialized Hashtable for the member table
```

```
136:    */
137:    private static java.util.Hashtable init() {
138:        Hashtable members = new Hashtable();
139:        members.put("1", VALUE_1);
140:        members.put("2", VALUE_2);
141:        members.put("3", VALUE_3);
142:        return members;
143:    }
144:
145:    /**
146:     * Method readResolve. will be called during deserialization to
147:     * replace the deserialized object with the correct constant
148:     * instance.
149:     *
150:     * @return this deserialized object
151:     */
152:    private java.lang.Object readResolve() {
153:        return valueOf(this.stringValue);
154:    }
155:
156:    /**
157:     * Method toString.Returns the String representation of this
158:     * ActivityStartReason
159:     *
160:     * @return the String representation of this ActivityStartReason
161:     */
162:    public java.lang.String toString() {
163:        return this.stringValue;
164:    }
165:
166:    /**
167:     * Method valueOf.Returns a new ActivityStartReason based on
168:     * the given String value.
169:     *
170:     * @param string
171:     * @return the ActivityStartReason value of parameter 'string'
172:     */
173:    public static ActivityStartReason valueOf(
174:        final java.lang.String string) {
175:        java.lang.Object obj = null;
176:        if (string != null) {
177:            obj = _memberTable.get(string);
178:        }
179:        if (obj == null) {
180:            String err = "" + string + " is not a valid ActivityStartReason";
```

```
181:         throw new IllegalArgumentException(err);
182:     }
183:     return (ActivityStartReason) obj;
184: }
185: /**
186:  * Method getInstance.
187:  * add by Vale
188:  */
189: public static Object getInstance(String string) {
190:     java.lang.Object obj = null;
191:     if (string != null) {
192:         obj = _memberTable.get(string);
193:     }
194:     if (obj == null) {
195:         String err = "" + string + " is not a valid ActivityStartReason";
196:         throw new IllegalArgumentException(err);
197:     }
198:     return (ActivityStartReason) obj;
199: }
200:
201: /**
202:  * Method setType. Set the type of this ActivityStartReason
203:  * add by Vale
204:  */
205: public void setType(int type) {
206:     this.type = type;
207: }
208:
209: /**
210:  * Method setStringValue. Set the stringValue of this ActivityStartReason
211:  * add by Vale
212:  */
213: public void setStringValue(String stringValue) {
214:     this.stringValue = stringValue;
215: }
216: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      ActivityStartReasonUserType.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30:
31: package export.types;
32:
33: import java.io.Serializable;
34: import java.sql.PreparedStatement;
35: import java.sql.ResultSet;
36: import java.sql.SQLException;
37: import java.sql.Types;
38:
39: import org.hibernate.HibernateException;
40: import org.hibernate.usertype.UserType;
41:
42: public class ActivityStartReasonUserType implements UserType {
43:
44:     private static final int[] SQL_TYPES = {Types.VARCHAR};
45:
```

```
46:     public int[] sqlTypes() {
47:         return SQL_TYPES;
48:     }
49:
50:     public Class returnedClass() {
51:         return ActivityStartReason.class;
52:     }
53:
54:     public boolean equals(Object x, Object y) {
55:         return x == y;
56:     }
57:
58:     public Object deepCopy(Object value) {
59:         return value;
60:     }
61:
62:     public boolean isMutable() {
63:         return false;
64:     }
65:
66:     public Object nullSafeGet(ResultSet resultSet, String[] names, Object owner) throws HibernateException,
SQLException {
67:         String name = resultSet.getString(names[0]);
68:         return resultSet.isNull() ? null : ActivityStartReason.getInstance(name);
69:     }
70:
71:     public void nullSafeSet(PreparedStatement statement, Object value, int index) throws HibernateException,
SQLException {
72:
73:         if (value == null) {
74:             statement.setNull(index, Types.VARCHAR);
75:         } else {
76:             statement.setString(index, value.toString());
77:         }
78:     }
79:
80:     public Object assemble(Serializable arg0, Object arg1) throws HibernateException {
81:         // TODO Auto-generated method stub
82:         return null;
83:     }
84:
85:     public Serializable disassemble(Object arg0) throws HibernateException {
86:         // TODO Auto-generated method stub
87:         return null;
88:     }
```



```
89:
90:     public int hashCode(Object arg0) throws HibernateException {
91:         // TODO Auto-generated method stub
92:         return 0;
93:     }
94:
95:     public Object replace(Object arg0, Object arg1, Object arg2) throws HibernateException {
96:         // TODO Auto-generated method stub
97:         return null;
98:     }
99: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      BIRODataset.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: /**
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36:
37: package export.types;
38:
39: //-----/
40: //- Imported classes and packages -/
41: //-----/
42:
43: import java.util.Hashtable;
44:
45: import export.types.BIRODataSet;
```

```
46:
47: /**
48:  * Class BIRODataSet.
49:  *
50:  * @version $Revision$ $Date$
51:  */
52: public class BIRODataSet implements java.io.Serializable {
53:
54:
55:     //-----/
56:     //- Class/Member Variables -/
57:     //-----/
58:
59:
60:     /**
61:      * The PAT_ID type
62:      */
63:     public static final String PAT_ID_TYPE = "BIRO001";
64:
65:     /**
66:      * The instance of the PAT_ID type
67:      */
68:     public static final BIRODataSet PAT_ID = new BIRODataSet(PAT_ID_TYPE, "PAT_ID");
69:
70:     /**
71:      * The DS_ID type
72:      */
73:     public static final String DS_ID_TYPE = "BIRO002";
74:
75:     /**
76:      * The instance of the DS_ID type
77:      */
78:     public static final BIRODataSet DS_ID = new BIRODataSet(DS_ID_TYPE, "DS_ID");
79:
80:     /**
81:      * The TYPE_DM type
82:      */
83:     public static final String TYPE_DM_TYPE = "BIRO003";
84:
85:     /**
86:      * The instance of the TYPE_DM type
87:      */
88:     public static final BIRODataSet TYPE_DM = new BIRODataSet(TYPE_DM_TYPE, "TYPE_DM");
89:
90:     /**
```

```
91:      * The SEX type
92:      */
93:      public static final String SEX_TYPE = "BIRO004";
94:
95:      /**
96:       * The instance of the SEX type
97:       */
98:      public static final BIRODataSet SEX = new BIRODataSet(SEX_TYPE, "SEX");
99:
100:     /**
101:      * The DOB type
102:      */
103:      public static final String DOB_TYPE = "BIRO005";
104:
105:      /**
106:       * The instance of the DOB type
107:       */
108:      public static final BIRODataSet DOB = new BIRODataSet(DOB_TYPE, "DOB");
109:
110:     /**
111:      * The DT_DIAG type
112:      */
113:      public static final String DT_DIAG_TYPE = "BIRO006";
114:
115:      /**
116:       * The instance of the DT_DIAG type
117:       */
118:      public static final BIRODataSet DT_DIAG = new BIRODataSet(DT_DIAG_TYPE, "DT_DIAG");
119:
120:     /**
121:      * The EPI_DATE type
122:      */
123:      public static final String EPI_DATE_TYPE = "BIRO007";
124:
125:      /**
126:       * The instance of the EPI_DATE type
127:       */
128:      public static final BIRODataSet EPI_DATE = new BIRODataSet(EPI_DATE_TYPE, "EPI_DATE");
129:
130:     /**
131:      * The SMOK_STAT type
132:      */
133:      public static final String SMOK_STAT_TYPE = "BIRO008";
134:
135:     /**
```

```
136:      * The instance of the SMOK_STAT type
137:      */
138:      public static final BIRODataSet SMOK_STAT = new BIRODataSet(SMOK_STAT_TYPE, "SMOK_STAT");
139:
140:      /**
141:       * The CIGS_DAY type
142:       */
143:      public static final String CIGS_DAY_TYPE = "BIRO009";
144:
145:      /**
146:       * The instance of the CIGS_DAY type
147:       */
148:      public static final BIRODataSet CIGS_DAY = new BIRODataSet(CIGS_DAY_TYPE, "CIGS_DAY");
149:
150:
151:      /**
152:       * The ALCOHOL type
153:       */
154:      public static final String ALCOHOL_TYPE = "BIRO010";
155:
156:      /**
157:       * The instance of the ALCOHOL type
158:       */
159:      public static final BIRODataSet ALCOHOL = new BIRODataSet(ALCOHOL_TYPE, "ALCOHOL");
160:
161:      /**
162:       * The WEIGHT type
163:       */
164:      public static final String WEIGHT_TYPE = "BIRO011";
165:
166:      /**
167:       * The instance of the WEIGHT type
168:       */
169:      public static final BIRODataSet WEIGHT = new BIRODataSet(WEIGHT_TYPE, "WEIGHT");
170:
171:      /**
172:       * The HEIGHT type
173:       */
174:      public static final String HEIGHT_TYPE = "BIRO012";
175:
176:      /**
177:       * The instance of the HEIGHT type
178:       */
179:      public static final BIRODataSet HEIGHT = new BIRODataSet(HEIGHT_TYPE, "HEIGHT");
180:
```

```
181:  /**
182:   * The BMI type
183:   */
184:  public static final String BMI_TYPE = "BIRO013";
185:
186:  /**
187:   * The instance of the BMI type
188:   */
189:  public static final BIRODataSet BMI = new BIRODataSet(BMI_TYPE, "BMI");
190:
191:  /**
192:   * The SBP type
193:   */
194:  public static final String SBP_TYPE = "BIRO014";
195:
196:  /**
197:   * The instance of the SBP type
198:   */
199:  public static final BIRODataSet SBP = new BIRODataSet(SBP_TYPE, "SBP");
200:
201:  /**
202:   * The DBP type
203:   */
204:  public static final String DBP_TYPE = "BIRO015";
205:
206:  /**
207:   * The instance of the DBP type
208:   */
209:  public static final BIRODataSet DBP = new BIRODataSet(DBP_TYPE, "DBP");
210:
211:  /**
212:   * The HBA1C type
213:   */
214:  public static final String HBA1C_TYPE = "BIRO016";
215:
216:  /**
217:   * The instance of the HBA1C type
218:   */
219:  public static final BIRODataSet HBA1C = new BIRODataSet(HBA1C_TYPE, "HBA1C");
220:
221:  /**
222:   * The CREAT type
223:   */
224:  public static final String CREAT_TYPE = "BIRO017";
225:
```

```
226:  /**
227:   * The instance of the CREAT type
228:   */
229:  public static final BIRODataSet CREAT = new BIRODataSet(CREAT_TYPE, "CREAT");
230:
231:  /**
232:   * The MA_TEST type
233:   */
234:  public static final String MA_TEST_TYPE = "BIRO018";
235:
236:  /**
237:   * The instance of the MA_TEST type
238:   */
239:  public static final BIRODataSet MA_TEST = new BIRODataSet(MA_TEST_TYPE, "MA_TEST");
240:
241:  /**
242:   * The CHOL type
243:   */
244:  public static final String CHOL_TYPE = "BIRO019";
245:
246:  /**
247:   * The instance of the CHOL type
248:   */
249:  public static final BIRODataSet CHOL = new BIRODataSet(CHOL_TYPE, "CHOL");
250:
251:  /**
252:   * The HDL type
253:   */
254:  public static final String HDL_TYPE = "BIRO020";
255:
256:  /**
257:   * The instance of the HDL type
258:   */
259:  public static final BIRODataSet HDL = new BIRODataSet(HDL_TYPE, "HDL");
260:
261:  /**
262:   * The LDL type
263:   */
264:  public static final String LDL_TYPE = "BIRO046";
265:
266:  /**
267:   * The instance of the LDL type
268:   */
269:  public static final BIRODataSet LDL = new BIRODataSet(LDL_TYPE, "LDL");
270:
```

```
271:  /**
272:   * The TG type
273:   */
274:  public static final String TG_TYPE = "BIRO021";
275:
276:  /**
277:   * The instance of the TG type
278:   */
279:  public static final BIRODataSet TG = new BIRODataSet(TG_TYPE, "TG");
280:
281:  /**
282:   * The RETINAL_EXAM type
283:   */
284:  public static final String RETINAL_EXAM_TYPE = "BIRO022";
285:
286:  /**
287:   * The instance of the RETINAL_EXAM type
288:   */
289:  public static final BIRODataSet RETINAL_EXAM = new BIRODataSet(RETINAL_EXAM_TYPE, "RETINAL_EXAM");
290:
291:  /**
292:   * The RETINA type
293:   */
294:  public static final String RETINA_TYPE = "BIRO023";
295:
296:  /**
297:   * The instance of the RETINA type
298:   */
299:  public static final BIRODataSet RETINA = new BIRODataSet(RETINA_TYPE, "RETINA");
300:
301:  /**
302:   * The MACULA type
303:   */
304:  public static final String MACULA_TYPE = "BIRO024";
305:
306:  /**
307:   * The instance of the MACULA type
308:   */
309:  public static final BIRODataSet MACULA = new BIRODataSet(MACULA_TYPE, "MACULA");
310:
311:  /**
312:   * The FOOT_EXAM type
313:   */
314:  public static final String FOOT_EXAM_TYPE = "BIRO025";
315:
```



```
316:  /**
317:   * The instance of the FOOT_EXAM type
318:   */
319:  public static final BIRODataSet FOOT_EXAM = new BIRODataSet(FOOT_EXAM_TYPE, "FOOT_EXAM");
320:
321:  /**
322:   * The PULSES type
323:   */
324:  public static final String PULSES_TYPE = "BIRO026";
325:
326:  /**
327:   * The instance of the PULSES type
328:   */
329:  public static final BIRODataSet PULSES = new BIRODataSet(PULSES_TYPE, "PULSES");
330:
331:  /**
332:   * The FTSENS type
333:   */
334:  public static final String FTSENS_TYPE = "BIRO027";
335:
336:  /**
337:   * The instance of the FTSENS type
338:   */
339:  public static final BIRODataSet FTSENS = new BIRODataSet(FTSENS_TYPE, "FTSENS");
340:
341:  /**
342:   * The ESRF type
343:   */
344:  public static final String ESRF_TYPE = "BIRO028";
345:
346:  /**
347:   * The instance of the ESRF type
348:   */
349:  public static final BIRODataSet ESRF = new BIRODataSet(ESRF_TYPE, "ESRF");
350:
351:  /**
352:   * The DIALYSIS type
353:   */
354:  public static final String DIALYSIS_TYPE = "BIRO029";
355:
356:  /**
357:   * The instance of the DIALYSIS type
358:   */
359:  public static final BIRODataSet DIALYSIS = new BIRODataSet(DIALYSIS_TYPE, "DIALYSIS");
360:
```

```
361:  /**
362:   * The TRANSPLANT type
363:   */
364:  public static final String TRANSPLANT_TYPE = "BIRO030";
365:
366:  /**
367:   * The instance of the TRANSPLANT type
368:   */
369:  public static final BIRODataSet TRANSPLANT = new BIRODataSet(TRANSPLANT_TYPE, "TRANSPLANT");
370:
371:  /**
372:   * The STROKE type
373:   */
374:  public static final String STROKE_TYPE = "BIRO031";
375:
376:  /**
377:   * The instance of the STROKE type
378:   */
379:  public static final BIRODataSet STROKE = new BIRODataSet(STROKE_TYPE, "STROKE");
380:
381:  /**
382:   * The ULCER type
383:   */
384:  public static final String ULCER_TYPE = "BIRO032";
385:
386:  /**
387:   * The instance of the ULCER type
388:   */
389:  public static final BIRODataSet ULCER = new BIRODataSet(ULCER_TYPE, "ULCER");
390:
391:  /**
392:   * The MI type
393:   */
394:  public static final String MI_TYPE = "BIRO033";
395:
396:  /**
397:   * The instance of the MI type
398:   */
399:  public static final BIRODataSet MI = new BIRODataSet(MI_TYPE, "MI");
400:
401:  /**
402:   * The LASER type
403:   */
404:  public static final String LASER_TYPE = "BIRO034";
405:
```

```
406:  /**
407:   * The instance of the LASER type
408:   */
409:  public static final BIRODataSet LASER = new BIRODataSet(LASER_TYPE, "LASER");
410:
411:  /**
412:   * The HYPERTENSION type
413:   */
414:  public static final String HYPERTENSION_TYPE = "BIRO035";
415:
416:  /**
417:   * The instance of the HYPERTENSION type
418:   */
419:  public static final BIRODataSet HYPERTENSION = new BIRODataSet(HYPERTENSION_TYPE, "HYPERTENSION");
420:
421:  /**
422:   * The BLIND type
423:   */
424:  public static final String BLIND_TYPE = "BIRO036";
425:
426:  /**
427:   * The instance of the BLIND type
428:   */
429:  public static final BIRODataSet BLIND = new BIRODataSet(BLIND_TYPE, "BLIND");
430:
431:  /**
432:   * The AMPUT type
433:   */
434:  public static final String AMPUT_TYPE = "BIRO037";
435:
436:  /**
437:   * The instance of the AMPUT type
438:   */
439:  public static final BIRODataSet AMPUT = new BIRODataSet(AMPUT_TYPE, "AMPUT");
440:
441:  /**
442:   * The HYPERT_MED type
443:   */
444:  public static final String HYPERT_MED_TYPE = "BIRO038";
445:
446:  /**
447:   * The instance of the HYPERT_MED type
448:   */
449:  public static final BIRODataSet HYPERT_MED = new BIRODataSet(HYPERT_MED_TYPE, "HYPERT_MED");
450:
```

```
451:  /**
452:   * The DRUG_THERAPY type
453:   */
454:  public static final String DRUG_THERAPY_TYPE = "BIRO039";
455:
456:  /**
457:   * The instance of the DRUG_THERAPY type
458:   */
459:  public static final BIRODataSet DRUG_THERAPY = new BIRODataSet(DRUG_THERAPY_TYPE, "DRUG_THERAPY");
460:
461:  /**
462:   * The ORAL_THERAPY type
463:   */
464:  public static final String ORAL_THERAPY_TYPE = "BIRO040";
465:
466:  /**
467:   * The instance of the ORAL_THERAPY type
468:   */
469:  public static final BIRODataSet ORAL_THERAPY = new BIRODataSet(ORAL_THERAPY_TYPE, "ORAL_THERAPY");
470:
471:  /**
472:   * The PUMP_THERAPY type
473:   */
474:  public static final String PUMP_THERAPY_TYPE = "BIRO041";
475:
476:  /**
477:   * The instance of the PUMP_THERAPY type
478:   */
479:  public static final BIRODataSet PUMP_THERAPY = new BIRODataSet(PUMP_THERAPY_TYPE, "PUMP_THERAPY");
480:
481:  /**
482:   * The NASAL_THERAPY type
483:   */
484:  public static final String NASAL_THERAPY_TYPE = "BIRO042";
485:
486:  /**
487:   * The instance of the NASAL_THERAPY type
488:   */
489:  public static final BIRODataSet NASAL_THERAPY = new BIRODataSet(NASAL_THERAPY_TYPE, "NASAL_THERAPY");
490:
491:  /**
492:   * The INJECTIONS type
493:   */
494:  public static final String INJECTIONS_TYPE = "BIRO043";
495:
```

```
496:  /**
497:   * The instance of the INJECTIONS type
498:   */
499:  public static final BIRODataSet INJECTIONS = new BIRODataSet(INJECTIONS_TYPE, "INJECTIONS");
500:
501:  /**
502:   * The SELF_MON type
503:   */
504:  public static final String SELF_MON_TYPE = "BIRO044";
505:
506:  /**
507:   * The instance of the SELF_MON type
508:   */
509:  public static final BIRODataSet SELF_MON = new BIRODataSet(SELF_MON_TYPE, "SELF_MON");
510:
511:  /**
512:   * The EDUCATION type
513:   */
514:  public static final String EDUCATION_TYPE = "BIRO045";
515:
516:  /**
517:   * The instance of the EDUCATION type
518:   */
519:  public static final BIRODataSet EDUCATION = new BIRODataSet(EDUCATION_TYPE, "EDUCATION");
520:
521:
522:  /**
523:   * The LIPID_THERAPY type
524:   */
525:  public static final String LIPID_THERAPY_TYPE = "BIRO053";
526:
527:  /**
528:   * The instance of the LIPID_THERAPY type
529:   */
530:  public static final BIRODataSet LIPID_THERAPY = new BIRODataSet(LIPID_THERAPY_TYPE, "LIPID_THERAPY");
531:
532:  /**
533:   * The ANTIPLATELET_THERAPY type
534:   */
535:  public static final String ANTIPLATELET_THERAPY_TYPE = "BIRO054";
536:
537:  /**
538:   * The instance of the ANTIPLATELET_THERAPY type
539:   */
540:  public static final BIRODataSet ANTIPLATELET_THERAPY = new BIRODataSet(ANTIPLATELET_THERAPY_TYPE,
```

```
"ANTIPLATELET_THERAPY");
541:
542:     /**
543:     * The DMP_ENROL type
544:     */
545:     public static final String ENROL_DMP_TYPE = "BIRO048";
546:
547:     /**
548:     * The instance of the DMP_ENROL type
549:     */
550:     public static final BIRODataSet ENROL_DMP = new BIRODataSet(ENROL_DMP_TYPE, "ENROL_DMP");
551:
552:     /**
553:     * The ALC_STAT type
554:     */
555:     public static final String ALC_STAT_TYPE = "BIRO047";
556:
557:     /**
558:     * The instance of the ALC_STAT type
559:     */
560:     public static final BIRODataSet ALC_STAT = new BIRODataSet(ALC_STAT_TYPE, "ALC_STAT");
561:
562:     /**
563:     * The AD_START_DATE type
564:     */
565:     public static final String AD_START_DATE_TYPE = "BIRO049";
566:
567:     /**
568:     * The instance of the AD_START_DATE type
569:     */
570:     public static final BIRODataSet AD_START_DATE = new BIRODataSet(AD_START_DATE_TYPE, "AD_START_DATE");
571:
572:     /**
573:     * The AD_START_REASON type
574:     */
575:     public static final String AD_START_REASON_TYPE = "BIRO050";
576:
577:     /**
578:     * The instance of the AD_START_REASON type
579:     */
580:     public static final BIRODataSet AD_START_REASON = new BIRODataSet(AD_START_REASON_TYPE, "AD_START_REASON");
581:
582:     /**
583:     * The AD_END_DATE type
584:
```

```
585:      */
586:      public static final String AD_END_DATE_TYPE = "BIRO051";
587:
588:      /**
589:       * The instance of the AD_END_DATE type
590:       */
591:      public static final BIRODataSet AD_END_DATE = new BIRODataSet(AD_END_DATE_TYPE, "AD_END_DATE");
592:
593:      /**
594:       * The AD_END_REASON type
595:       */
596:      public static final String AD_END_REASON_TYPE = "BIRO052";
597:
598:      /**
599:       * The instance of the AD_END_REASON type
600:       */
601:      public static final BIRODataSet AD_END_REASON = new BIRODataSet(AD_END_REASON_TYPE, "AD_END_REASON");
602:
603:
604:      /**
605:       * Field _memberTable.
606:       */
607:      private static java.util.Hashtable _memberTable = init();
608:
609:      /**
610:       * Field type.
611:       */
612:      private String type = null;
613:
614:      /**
615:       * Field stringValue.
616:       */
617:      private java.lang.String stringValue = null;
618:
619:
620:      //-----/
621:      //- Constructors -/
622:      //-----/
623:      //add by Vale
624:      public BIRODataSet() {
625:          super();
626:          this.type = null;
627:          this.stringValue = null;
628:      }
629:
```

```
630: private BIRODataSet(final java.lang.String type, final java.lang.String value) {
631:     super();
632:     this.type = type;
633:     this.stringValue = value;
634: }
635:
636:
637:     //-----/
638:     //- Methods -/
639:     //-----/
640:
641: /**
642:  * Method enumerate.Returns an enumeration of all possible
643:  * instances of BIRODataSet
644:  *
645:  * @return an Enumeration over all possible instances of
646:  * BIRODataSet
647:  */
648: public static java.util.Enumeration enumerate(
649: ) {
650:     return _memberTable.elements();
651: }
652:
653: /**
654:  * Method getType.Returns the type of this BIRODataSet
655:  *
656:  * @return the type of this BIRODataSet
657:  */
658: public String getType(
659: ) {
660:     return this.type;
661: }
662:
663: /**
664:  * Method getStringValue.Returns the stringValue of this BIRODataSet
665:  *
666:  * @return the stringValue of this BIRODataSet
667:  */
668: public String getStringValue(
669: ) {
670:     return this.stringValue;
671: }
672:
673: /**
674:  * Method init.
```



```
675:      *
676:      * @return the initialized Hashtable for the member table
677:      */
678:  private static java.util.Hashtable init(
679:  ) {
680:      Hashtable members = new Hashtable();
681:      members.put("PAT_ID", PAT_ID);
682:      members.put("DS_ID", DS_ID);
683:      members.put("TYPE_DM", TYPE_DM);
684:      members.put("SEX", SEX);
685:      members.put("DOB", DOB);
686:      members.put("DT_DIAG", DT_DIAG);
687:      members.put("EPI_DATE", EPI_DATE);
688:      members.put("SMOK_STAT", SMOK_STAT);
689:      members.put("CIGS_DAY", CIGS_DAY);
690:      members.put("ALCOHOL", ALCOHOL);
691:      members.put("WEIGHT", WEIGHT);
692:      members.put("HEIGHT", HEIGHT);
693:      members.put("BMI", BMI);
694:      members.put("SBP", SBP);
695:      members.put("DBP", DBP);
696:      members.put("HBA1C", HBA1C);
697:      members.put("CREAT", CREAT);
698:      members.put("MA_TEST", MA_TEST);
699:      members.put("CHOL", CHOL);
700:      members.put("HDL", HDL);
701:      members.put("LDL", LDL);
702:      members.put("TG", TG);
703:      members.put("RETINAL_EXAM", RETINAL_EXAM);
704:      members.put("RETINA", RETINA);
705:      members.put("MACULA", MACULA);
706:      members.put("FOOT_EXAM", FOOT_EXAM);
707:      members.put("PULSES", PULSES);
708:      members.put("FTSENS", FTSENS);
709:      members.put("ESRF", ESRF);
710:      members.put("DIALYSIS", DIALYSIS);
711:      members.put("TRANSPLANT", TRANSPLANT);
712:      members.put("STROKE", STROKE);
713:      members.put("ULCER", ULCER);
714:      members.put("MI", MI);
715:      members.put("LASER", LASER);
716:      members.put("HYPERTENSION", HYPERTENSION);
717:      members.put("BLIND", BLIND);
718:      members.put("AMPUT", AMPUT);
719:      members.put("HYPERT_MED", HYPERT_MED);
```

```
720:         members.put("DRUG_THERAPY", DRUG_THERAPY);
721:         members.put("ORAL_THERAPY", ORAL_THERAPY);
722:         members.put("PUMP_THERAPY", PUMP_THERAPY);
723:         members.put("NASAL_THERAPY", NASAL_THERAPY);
724:         members.put("INJECTIONS", INJECTIONS);
725:         members.put("SELF_MON", SELF_MON);
726:         members.put("EDUCATION", EDUCATION);
727:         members.put("LIPID_THERAPY", LIPID_THERAPY);
728:         members.put("ANTIPLATELET_THERAPY", ANTIPLATELET_THERAPY);
729:         members.put("ENROL_DMP", ENROL_DMP);
730:         members.put("ALC_STAT", ALC_STAT);
731:         members.put("AD_START_DATE", AD_START_DATE);
732:         members.put("AD_START_REASON", AD_START_REASON);
733:         members.put("AD_END_DATE", AD_END_DATE);
734:         members.put("AD_END_REASON", AD_END_REASON);
735:
736:         return members;
737:     }
738:
739:     /**
740:      * Method readResolve. will be called during deserialization to
741:      * replace the deserialized object with the correct constant
742:      * instance.
743:      *
744:      * @return this deserialized object
745:      */
746:     private java.lang.Object readResolve(
747:     ) {
748:         return valueOf(this.stringValue);
749:     }
750:
751:     /**
752:      * Method toString.Returns the String representation of this
753:      * BIRODataSet
754:      *
755:      * @return the String representation of this BIRODataSet
756:      */
757:     public java.lang.String toString(
758:     ) {
759:         return this.stringValue;
760:     }
761:
762:     /**
763:      * Method valueOf.Returns a new BIRODataSet based on the given
764:      * String value.
```

```
765:      *
766:      * @param string
767:      * @return the BIRODataSet value of parameter 'string'
768:      */
769:      public static export.types.BIRODataSet valueOf(
770:          final java.lang.String string) {
771:          java.lang.Object obj = null;
772:          if (string != null) {
773:              obj = _memberTable.get(string);
774:          }
775:          if (obj == null) {
776:              String err = "" + string + " is not a valid BIRODataSet";
777:              throw new IllegalArgumentException(err);
778:          }
779:          return (BIRODataSet) obj;
780:      }
781:
782:
783:      public static Object getInstance(String string) {
784:          java.lang.Object obj = null;
785:          if (string != null) {
786:              obj = _memberTable.get(string);
787:          }
788:          if (obj == null) {
789:              String err = "" + string + " is not a valid BIRODataSet";
790:              throw new IllegalArgumentException(err);
791:          }
792:          return (BIRODataSet) obj;
793:      }
794:
795:      /**
796:      * Method setType. Set the type of this BIRODataSet
797:      * add by Vale
798:      */
799:      public void setType(String type){
800:          this.type=type;
801:      }
802:
803:      /**
804:      * Method setStringValue.Set the stringValue of this BIRODataSet
805:      * add by Vale
806:      */
807:      public void setStringValue(String stringValue) {
808:          this.stringValue = stringValue;
809:      }
```

```
810:
811: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      BIRODatasetUserType.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30:
31: package export.types;
32:
33: import java.io.Serializable;
34: import java.sql.PreparedStatement;
35: import java.sql.ResultSet;
36: import java.sql.SQLException;
37: import java.sql.Types;
38:
39: import org.hibernate.HibernateException;
40: import org.hibernate.usertype.UserType;
41:
42: public class BIRODataSetUserType implements UserType {
43:
44:     private static final int[] SQL_TYPES = {Types.VARCHAR};
45:
```

```
46: public int[] sqlTypes() { return SQL_TYPES; }
47: public Class returnedClass() { return BIRODataSet.class; }
48: public boolean equals(Object x, Object y) { return x == y; }
49: public Object deepCopy(Object value) { return value; }
50: public boolean isMutable() { return false; }
51:
52: public Object nullSafeGet(ResultSet resultSet,
53:                           String[] names,
54:                           Object owner)
55:     throws HibernateException, SQLException {
56:
57:     String name = resultSet.getString(names[0]);
58:     return resultSet.isNull() ? null : BIRODataSet.getInstance(name);
59: }
60:
61: public void nullSafeSet(PreparedStatement statement,
62:                         Object value,
63:                         int index)
64:     throws HibernateException, SQLException {
65:
66:     if (value == null) {
67:         statement.setNull(index, Types.VARCHAR);
68:     } else {
69:         statement.setString(index, value.toString());
70:     }
71: }
72: public Object assemble(Serializable arg0, Object arg1) throws HibernateException {
73:     // TODO Auto-generated method stub
74:     return null;
75: }
76: public Serializable disassemble(Object arg0) throws HibernateException {
77:     // TODO Auto-generated method stub
78:     return null;
79: }
80: public int hashCode(Object arg0) throws HibernateException {
81:     // TODO Auto-generated method stub
82:     return 0;
83: }
84: public Object replace(Object arg0, Object arg1, Object arg2) throws HibernateException {
85:     // TODO Auto-generated method stub
86:     return null;
87: }
88: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      DataSource.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10: * the Free Software Foundation; either version 2, or (at your option)
11: * any later version.
12: *
13: * This file is distributed in the hope that it will be useful,
14: * but WITHOUT ANY WARRANTY; without even the implied warranty of
15: * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16: * GNU General Public License for more details.
17: *
18: * You should have received a copy of the GNU General Public License
19: * along with this file; see the file COPYING. If not, write to
20: * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21: *
22: * In short: you may use this file any way you like, as long as you
23: * don't charge money for it, remove this notice, or hold anyone liable
24: * for its results.
25: *
26:
27: * GPL Copyright, The BIRO Project
28: *
29: **/
30: /*
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36:
37: package export.types;
38:
39: //-----/
40: //- Imported classes and packages -/
41: //-----/
42:
43: import java.util.Hashtable;
44:
45: /**
```

```
46:  * Class DataSource.
47:  *
48:  * @version $Revision$ $Date$
49:  */
50: public class DataSource implements java.io.Serializable {
51:
52:
53:     //-----/
54:     //- Class/Member Variables -/
55:     //-----/
56:
57:     /**
58:      * The 1 type
59:      */
60:     public static final int VALUE_1_TYPE = 0;
61:
62:     /**
63:      * The instance of the 1 type
64:      */
65:     public static final DataSource VALUE_1 = new DataSource(VALUE_1_TYPE, "1");
66:
67:     /**
68:      * The 2 type
69:      */
70:     public static final int VALUE_2_TYPE = 1;
71:
72:     /**
73:      * The instance of the 2 type
74:      */
75:     public static final DataSource VALUE_2 = new DataSource(VALUE_2_TYPE, "2");
76: /**
77:      * The 3 type
78:      */
79:     public static final int VALUE_3_TYPE = 2;
80:
81:     /**
82:      * The instance of the 3 type
83:      */
84:     public static final DataSource VALUE_3 = new DataSource(VALUE_3_TYPE, "3");
85: /**
86:      * The 4 type
87:      */
88:     public static final int VALUE_4_TYPE = 3;
89:
90:     /**
```



```
91:      * The instance of the 4 type
92:      */
93:      public static final DataSource VALUE_4 = new DataSource(VALUE_4_TYPE, "4");
94:      /**
95:       * The 5 type
96:       */
97:      public static final int VALUE_5_TYPE = 4;
98:
99:      /**
100:       * The instance of the 5 type
101:       */
102:      public static final DataSource VALUE_5 = new DataSource(VALUE_5_TYPE, "5");
103:      /**
104:       * The 6 type
105:       */
106:      public static final int VALUE_6_TYPE = 5;
107:
108:      /**
109:       * The instance of the 6 type
110:       */
111:      public static final DataSource VALUE_6 = new DataSource(VALUE_6_TYPE, "6");
112:      /**
113:       * The 7 type
114:       */
115:      public static final int VALUE_7_TYPE = 6;
116:
117:      /**
118:       * The instance of the 7 type
119:       */
120:      public static final DataSource VALUE_7 = new DataSource(VALUE_7_TYPE, "7");
121:      /**
122:       * Field _memberTable.
123:       */
124:      private static java.util.Hashtable _memberTable = init();
125:
126:      /**
127:       * Field type.
128:       */
129:      private int type = -1;
130:
131:      /**
132:       * Field stringValue.
133:       */
134:      private java.lang.String stringValue = null;
135:
```

```
136:
137:     //-----/
138:     //- Constructors -/
139:     //-----/
140:     //add by Valentina
141:     public DataSource() {
142:         super();
143:         this.type = -1;
144:         this.stringValue = null;
145:     }
146:
147:     private DataSource(final int type, final java.lang.String value) {
148:         super();
149:         this.type = type;
150:         this.stringValue = value;
151:     }
152:
153:
154:     //-----/
155:     //- Methods -/
156:     //-----/
157:
158:     /**
159:      * Method enumerate.Returns an enumeration of all possible
160:      * instances of DataSource
161:      *
162:      * @return an Enumeration over all possible instances of
163:      * DataSource
164:      */
165:     public static java.util Enumeration enumerate(
166:     ) {
167:         return _memberTable.elements();
168:     }
169:
170:     /**
171:      * Method getType.Returns the type of this DataSource
172:      *
173:      * @return the type of this DataSource
174:      */
175:     public int getType(
176:     ) {
177:         return this.type;
178:     }
179:
180:
```

```
181:  /**Aggiunto da Valentina
182:   * Method getStringValue.Returns the stringValue of this DataSource
183:   *
184:   * @return the stringValue of this DataSource
185:   */
186:  public String getStringValue(
187:  ) {
188:      return this.stringValue;
189:  }
190:
191:
192:  /**
193:   * Method init.
194:   *
195:   * @return the initialized Hashtable for the member table
196:   */
197:  private static java.util.Hashtable init(
198:  ) {
199:      Hashtable members = new Hashtable();
200:      members.put("1", VALUE_1);
201:      members.put("2", VALUE_2);
202:      members.put("3", VALUE_3);
203:      members.put("4", VALUE_4);
204:      members.put("5", VALUE_5);
205:      members.put("6", VALUE_6);
206:      members.put("7", VALUE_7);
207:      return members;
208:  }
209:
210:  /**
211:   * Method readResolve. will be called during deserialization to
212:   * replace the deserialized object with the correct constant
213:   * instance.
214:   *
215:   * @return this deserialized object
216:   */
217:  private java.lang.Object readResolve(
218:  ) {
219:      return valueOf(this.stringValue);
220:  }
221:
222:  /**
223:   * Method toString.Returns the String representation of this
224:   * DataSource
225:   *
```

```
226:      * @return the String representation of this DataSource
227:      */
228: public java.lang.String toString(
229: ) {
230:     return this.stringValue;
231: }
232:
233: /**
234:  * Method valueOf.Returns a new DataSource based on the given
235:  * String value.
236:  *
237:  * @param string
238:  * @return the DataSource value of parameter 'string'
239:  */
240: public static export.types.DataSource valueOf(
241:     final java.lang.String string) {
242:     java.lang.Object obj = null;
243:     if (string != null) {
244:         obj = _memberTable.get(string);
245:     }
246:     if (obj == null) {
247:         String err = "" + string + " is not a valid DataSource";
248:         throw new IllegalArgumentException(err);
249:     }
250:     return (DataSource) obj;
251: }
252:
253:
254: /**
255:  * Method getInstance.
256:  * add by Vale
257:  */
258: public static Object getInstance(String string) {
259:     java.lang.Object obj = null;
260:     if (string != null) {
261:         obj = _memberTable.get(string);
262:     }
263:     if (obj == null) {
264:         String err = "" + string + " is not a valid DataSource";
265:         throw new IllegalArgumentException(err);
266:     }
267:     return (DataSource) obj;
268: }
269:
270: /**
```

```
271:      * Method setType. Set the type of this DataSource
272:      * add by Vale
273:      */
274:  public void setType(int type){
275:      this.type=type;
276:  }
277:
278:  /**
279:      * Method setStringValue.Set the stringValue of this DataSource
280:      * add by Vale
281:      */
282:  public void setStringValue(String stringValue) {
283:      this.stringValue = stringValue;
284:  }
285:
286: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      DataSourceUserType.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30:
31: package export.types;
32:
33: import java.io.Serializable;
34: import java.sql.PreparedStatement;
35: import java.sql.ResultSet;
36: import java.sql.SQLException;
37: import java.sql.Types;
38:
39: import org.hibernate.HibernateException;
40: import org.hibernate.usertype.UserType;
41:
42: public class DataSourceUserType implements UserType {
43:
44:     private static final int[] SQL_TYPES = {Types.VARCHAR};
45:
```

```
46: public int[] sqlTypes() { return SQL_TYPES; }
47: public Class returnedClass() { return DataSource.class; }
48: public boolean equals(Object x, Object y) { return x == y; }
49: public Object deepCopy(Object value) { return value; }
50: public boolean isMutable() { return false; }
51:
52: public Object nullSafeGet(ResultSet resultSet,
53:                           String[] names,
54:                           Object owner)
55:     throws HibernateException, SQLException {
56:
57:     String name = resultSet.getString(names[0]);
58:     return resultSet.isNull() ? null : DataSource.getInstance(name);
59: }
60:
61: public void nullSafeSet(PreparedStatement statement,
62:                        Object value,
63:                        int index)
64:     throws HibernateException, SQLException {
65:
66:     if (value == null) {
67:         statement.setNull(index, Types.VARCHAR);
68:     } else {
69:         statement.setString(index, value.toString());
70:     }
71: }
72: public Object assemble(Serializable arg0, Object arg1) throws HibernateException {
73:     // TODO Auto-generated method stub
74:     return null;
75: }
76: public Serializable disassemble(Object arg0) throws HibernateException {
77:     // TODO Auto-generated method stub
78:     return null;
79: }
80: public int hashCode(Object arg0) throws HibernateException {
81:     // TODO Auto-generated method stub
82:     return 0;
83: }
84: public Object replace(Object arg0, Object arg1, Object arg2) throws HibernateException {
85:     // TODO Auto-generated method stub
86:     return null;
87: }
88: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      Ranking.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30:
31: /*
32:  * This class was automatically generated with
33:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
34:  * Schema.
35:  * $Id$
36:  */
37:
38: package export.types;
39:
40:  //-----/
41:  //- Imported classes and packages -/
42:  //-----/
43:
44: import java.util.Hashtable;
45:
```



```
46: /**
47:  * Class Ranking.
48:  *
49:  * @version $Revision$ $Date$
50:  */
51: public class Ranking implements java.io.Serializable {
52:
53:
54:     //-----/
55:     //- Class/Member Variables -/
56:     //-----/
57:
58:     /**
59:      * The Low type
60:      */
61:     public static final int LOW_TYPE = 0;
62:
63:     /**
64:      * The instance of the Low type
65:      */
66:     public static final Ranking LOW = new Ranking(LOW_TYPE, "LOW");
67:
68:     /**
69:      * The Medium type
70:      */
71:     public static final int MEDIUM_TYPE = 1;
72:
73:     /**
74:      * The instance of the Medium type
75:      */
76:     public static final Ranking MEDIUM = new Ranking(MEDIUM_TYPE, "MEDIUM");
77:
78:     /**
79:      * The High type
80:      */
81:     public static final int HIGH_TYPE = 2;
82:
83:     /**
84:      * The instance of the High type
85:      */
86:     public static final Ranking HIGH = new Ranking(HIGH_TYPE, "HIGH");
87:
88:     /**
89:      * Field _memberTable.
90:      */
```

```
91: private static java.util.Hashtable _memberTable = init();
92:
93: /**
94:  * Field type.
95:  */
96: private int type = -1;
97:
98: /**
99:  * Field stringValue.
100:  */
101: private java.lang.String stringValue = null;
102:
103:
104:     //-----/
105:     //- Constructors -/
106:     //-----/
107:     //add by Valentina
108:     public Ranking() {
109:         super();
110:         this.type = -1;
111:         this.stringValue = null;
112:     }
113:
114:     private Ranking(final int type, final java.lang.String value) {
115:         super();
116:         this.type = type;
117:         this.stringValue = value;
118:     }
119:
120:
121:     //-----/
122:     //- Methods -/
123:     //-----/
124:
125:     /**
126:     * Method enumerate.Returns an enumeration of all possible
127:     * instances of Ranking
128:     *
129:     * @return an Enumeration over all possible instances of Ranking
130:     */
131:     public static java.util Enumeration enumerate(
132:     ) {
133:         return _memberTable.elements();
134:     }
135:
```

```
136:  /**
137:   * Method getType.Returns the type of this Ranking
138:   *
139:   * @return the type of this Ranking
140:   */
141:  public int getType(
142:  ) {
143:      return this.type;
144:  }
145:
146:  /**
147:   * Method getStringValue.Returns the stringValue of this Ranking
148:   *
149:   * @return the stringValue of this Ranking
150:   */
151:  public String getStringValue(
152:  ) {
153:      return this.stringValue;
154:  }
155:
156:  /**
157:   * Method init.
158:   *
159:   * @return the initialized Hashtable for the member table
160:   */
161:  private static java.util.Hashtable init(
162:  ) {
163:      Hashtable members = new Hashtable();
164:      members.put("LOW", LOW);
165:      members.put("MEDIUM", MEDIUM);
166:      members.put("HIGH", HIGH);
167:      return members;
168:  }
169:
170:  /**
171:   * Method readResolve. will be called during deserialization to
172:   * replace the deserialized object with the correct constant
173:   * instance.
174:   *
175:   * @return this deserialized object
176:   */
177:  private java.lang.Object readResolve(
178:  ) {
179:      return valueOf(this.stringValue);
180:  }
```

```
181:  /**
182:   * Method toString.Returns the String representation of this
183:   * Ranking
184:   *
185:   * @return the String representation of this Ranking
186:   */
187: public java.lang.String toString(
188: ) {
189:     return this.stringValue;
190: }
191:
192:  /**
193:   * Method valueOf.Returns a new Ranking based on the given
194:   * String value.
195:   *
196:   * @param string
197:   * @return the Ranking value of parameter 'string'
198:   */
199: public static export.types.Ranking valueOf(
200:     final java.lang.String string) {
201:     java.lang.Object obj = null;
202:     if (string != null) {
203:         obj = _memberTable.get(string);
204:     }
205:     if (obj == null) {
206:         String err = "" + string + " is not a valid Ranking";
207:         throw new IllegalArgumentException(err);
208:     }
209:     return (Ranking) obj;
210: }
211:
212:  /**
213:   * Method getInstance.
214:   * add by Vale
215:   */
216: public static Object getInstance(String string) {
217:     java.lang.Object obj = null;
218:     if (string != null) {
219:         obj = _memberTable.get(string);
220:     }
221:     if (obj == null) {
222:         String err = "" + string + " is not a valid Ranking";
223:         throw new IllegalArgumentException(err);
224:     }
225:     return (Ranking) obj;
```

```
226:         }
227:
228:         /**
229:      * Method setType. Set the type of this Ranking
230:      * add by Vale
231:      */
232:     public void setType(int type){
233:         this.type=type;
234:     }
235:
236:     /**
237:      * Method setStringValue.Set the stringValue of this Ranking
238:      * add by Vale
239:      */
240:     public void setStringValue(String stringValue) {
241:         this.stringValue = stringValue;
242:     }
243:
244: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      RankingUserType.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30:
31: package export.types;
32:
33: import java.io.Serializable;
34: import java.sql.PreparedStatement;
35: import java.sql.ResultSet;
36: import java.sql.SQLException;
37: import java.sql.Types;
38:
39: import org.hibernate.HibernateException;
40: import org.hibernate.usertype.UserType;
41:
42: public class RankingUserType implements UserType {
43:
44:     private static final int[] SQL_TYPES = {Types.VARCHAR};
45:
```

```
46: public int[] sqlTypes() { return SQL_TYPES; }
47: public Class returnedClass() { return Ranking.class; }
48: public boolean equals(Object x, Object y) { return x == y; }
49: public Object deepCopy(Object value) { return value; }
50: public boolean isMutable() { return false; }
51:
52: public Object nullSafeGet(ResultSet resultSet,
53:                           String[] names,
54:                           Object owner)
55:     throws HibernateException, SQLException {
56:
57:     String name = resultSet.getString(names[0]);
58:     return resultSet.isNull() ? null : Ranking.getInstance(name);
59: }
60:
61: public void nullSafeSet(PreparedStatement statement,
62:                        Object value,
63:                        int index)
64:     throws HibernateException, SQLException {
65:
66:     if (value == null) {
67:         statement.setNull(index, Types.VARCHAR);
68:     } else {
69:         statement.setString(index, value.toString());
70:     }
71: }
72: public Object assemble(Serializable arg0, Object arg1) throws HibernateException {
73:     // TODO Auto-generated method stub
74:     return null;
75: }
76: public Serializable disassemble(Object arg0) throws HibernateException {
77:     // TODO Auto-generated method stub
78:     return null;
79: }
80: public int hashCode(Object arg0) throws HibernateException {
81:     // TODO Auto-generated method stub
82:     return 0;
83: }
84: public Object replace(Object arg0, Object arg1, Object arg2) throws HibernateException {
85:     // TODO Auto-generated method stub
86:     return null;
87: }
88: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      SiteType.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30:
31: /*
32:  * This class was automatically generated with
33:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
34:  * Schema.
35:  * $Id$
36:  */
37:
38: package export.types;
39:
40:  //-----/
41:  //- Imported classes and packages -/
42:  //-----/
43:
44: import java.util.Hashtable;
45:
```



```
46: /**
47:  * The type of source from which data has been extracted
48:  *
49:  * @version $Revision$ $Date$
50:  */
51: public class SiteType implements java.io.Serializable {
52:
53:
54:     //-----/
55:     //- Class/Member Variables -/
56:     //-----/
57:
58:     /**
59:      * The 1 type
60:      */
61:     public static final int VALUE_1_TYPE = 0;
62:
63:     /**
64:      * The instance of the 1 type
65:      */
66:     public static final SiteType VALUE_1 = new SiteType(VALUE_1_TYPE, "1");
67:
68:     /**
69:      * The 2 type
70:      */
71:     public static final int VALUE_2_TYPE = 1;
72:
73:     /**
74:      * The instance of the 2 type
75:      */
76:     public static final SiteType VALUE_2 = new SiteType(VALUE_2_TYPE, "2");
77:
78:     /**
79:      * The 3 type
80:      */
81:     public static final int VALUE_3_TYPE = 2;
82:
83:     /**
84:      * The instance of the 3 type
85:      */
86:     public static final SiteType VALUE_3 = new SiteType(VALUE_3_TYPE, "3");
87:
88:     /**
89:      * The 4 type
90:      */
```

```
91:     public static final int VALUE_4_TYPE = 3;
92:
93:     /**
94:      * The instance of the 4 type
95:      */
96:     public static final SiteType VALUE_4 = new SiteType(VALUE_4_TYPE, "4");
97:
98:     /**
99:      * The 5 type
100:      */
101:     public static final int VALUE_5_TYPE = 4;
102:
103:     /**
104:      * The instance of the 5 type
105:      */
106:     public static final SiteType VALUE_5 = new SiteType(VALUE_5_TYPE, "5");
107:
108:     /**
109:      * The 6 type
110:      */
111:     public static final int VALUE_6_TYPE = 5;
112:
113:     /**
114:      * The instance of the 6 type
115:      */
116:     public static final SiteType VALUE_6 = new SiteType(VALUE_6_TYPE, "6");
117:
118:     /**
119:      * The 7 type
120:      */
121:     public static final int VALUE_7_TYPE = 6;
122:
123:     /**
124:      * The instance of the 7 type
125:      */
126:     public static final SiteType VALUE_7 = new SiteType(VALUE_7_TYPE, "7");
127:
128:     /**
129:      * The 8 type
130:      */
131:     public static final int VALUE_8_TYPE = 7;
132:
133:     /**
134:      * The instance of the 8 type
135:      */
```

```
136: public static final SiteType VALUE_8 = new SiteType(VALUE_8_TYPE, "8");
137:
138: /**
139:  * The 9 type
140:  */
141: public static final int VALUE_9_TYPE = 8;
142:
143: /**
144:  * The instance of the 9 type
145:  */
146: public static final SiteType VALUE_9 = new SiteType(VALUE_9_TYPE, "9");
147:
148: /**
149:  * The 10 type
150:  */
151: public static final int VALUE_10_TYPE = 9;
152:
153: /**
154:  * The instance of the 10 type
155:  */
156: public static final SiteType VALUE_10 = new SiteType(VALUE_10_TYPE, "10");
157:
158: /**
159:  * The 11 type
160:  */
161: public static final int VALUE_11_TYPE = 10;
162:
163: /**
164:  * The instance of the 11 type
165:  */
166: public static final SiteType VALUE_11 = new SiteType(VALUE_11_TYPE, "11");
167:
168: /**
169:  * The 12 type
170:  */
171: public static final int VALUE_12_TYPE = 11;
172:
173: /**
174:  * The instance of the 12 type
175:  */
176: public static final SiteType VALUE_12 = new SiteType(VALUE_12_TYPE, "12");
177:
178: /**
179:  * The 13 type
180:  */
```

```
181: public static final int VALUE_13_TYPE = 12;
182:
183: /**
184:  * The instance of the 13 type
185:  */
186: public static final SiteType VALUE_13 = new SiteType(VALUE_13_TYPE, "13");
187:
188: /**
189:  * Field _memberTable.
190:  */
191: private static java.util.Hashtable _memberTable = init();
192:
193: /**
194:  * Field type.
195:  */
196: private int type = -1;
197:
198: /**
199:  * Field stringValue.
200:  */
201: private java.lang.String stringValue = null;
202:
203:
204: //-----/
205: //- Constructors -/
206: //-----/
207: //add by Valentina
208: public SiteType() {
209:     super();
210:     this.type = -1;
211:     this.stringValue = null;
212: }
213:
214: private SiteType(final int type, final java.lang.String value) {
215:     super();
216:     this.type = type;
217:     this.stringValue = value;
218: }
219:
220:
221: //-----/
222: //- Methods -/
223: //-----/
224:
225: /**
```

```
226:      * Method enumerate.Returns an enumeration of all possible
227:      * instances of SiteType
228:      *
229:      * @return an Enumeration over all possible instances of SiteTyp
230:      */
231: public static java.util.Enumeration enumerate(
232: ) {
233:     return _memberTable.elements();
234: }
235:
236: /**
237:  * Method getType.Returns the type of this SiteType
238:  *
239:  * @return the type of this SiteType
240:  */
241: public int getType(
242: ) {
243:     return this.type;
244: }
245:
246:
247:
248: /**Aggiunto da Valentina
249:  * Method getStringValue.Returns the stringValue of this DataSource
250:  *
251:  * @return the stringValue of this DataSource
252:  */
253: public String getStringValue(
254: ) {
255:     return this.stringValue;
256: }
257:
258:
259: /**
260:  * Method init.
261:  *
262:  * @return the initialized Hashtable for the member table
263:  */
264: private static java.util.Hashtable init(
265: ) {
266:     Hashtable members = new Hashtable();
267:     members.put("1", VALUE_1);
268:     members.put("2", VALUE_2);
269:     members.put("3", VALUE_3);
270:     members.put("4", VALUE_4);
```

```
271:         members.put("5", VALUE_5);
272:         members.put("6", VALUE_6);
273:         members.put("7", VALUE_7);
274:         members.put("8", VALUE_8);
275:         members.put("9", VALUE_9);
276:         members.put("10", VALUE_10);
277:         members.put("11", VALUE_11);
278:         members.put("12", VALUE_12);
279:         members.put("13", VALUE_13);
280:         return members;
281:     }
282:
283:     /**
284:      * Method readResolve. will be called during deserialization to
285:      * replace the deserialized object with the correct constant
286:      * instance.
287:      *
288:      * @return this deserialized object
289:      */
290:     private java.lang.Object readResolve(
291:     ) {
292:         return valueOf(this.stringValue);
293:     }
294:
295:     /**
296:      * Method toString.Returns the String representation of this
297:      * SiteType
298:      *
299:      * @return the String representation of this SiteType
300:      */
301:     public java.lang.String toString(
302:     ) {
303:         return this.stringValue;
304:     }
305:
306:     /**
307:      * Method valueOf.Returns a new SiteType based on the given
308:      * String value.
309:      *
310:      * @param string
311:      * @return the SiteType value of parameter 'string'
312:      */
313:     public static export.types.SiteType valueOf(
314:         final java.lang.String string) {
315:         java.lang.Object obj = null;
```

```
316:         if (string != null) {
317:             obj = _memberTable.get(string);
318:         }
319:         if (obj == null) {
320:             String err = "" + string + " is not a valid SiteType";
321:             throw new IllegalArgumentException(err);
322:         }
323:         return (SiteType) obj;
324:     }
325:
326:     /**
327:     * Method getInstance.
328:     * add by Vale
329:     */
330:     public static Object getInstance(String string) {
331:         java.lang.Object obj = null;
332:         if (string != null) {
333:             obj = _memberTable.get(string);
334:         }
335:         if (obj == null) {
336:             String err = "" + string + " is not a valid SiteType";
337:             throw new IllegalArgumentException(err);
338:         }
339:         return (SiteType) obj;
340:     }
341:
342:     /**
343:     * Method setType. Set the type of this SiteType
344:     * add by Vale
345:     */
346:     public void setType(int type){
347:         this.type=type;
348:     }
349:
350:     /**
351:     * Method setStringValue.Set the stringValue of this SiteType
352:     * add by Vale
353:     */
354:     public void setStringValue(String stringValue) {
355:         this.stringValue = stringValue;
356:     }
357:
358:
359: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      SiteTypeUserType.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30:
31: package export.types;
32:
33: import java.io.Serializable;
34: import java.sql.PreparedStatement;
35: import java.sql.ResultSet;
36: import java.sql.SQLException;
37: import java.sql.Types;
38:
39: import org.hibernate.HibernateException;
40: import org.hibernate.usertype.UserType;
41:
42: public class SiteTypeUserType implements UserType {
43:
44:     private static final int[] SQL_TYPES = {Types.VARCHAR};
45:
```



```
46: public int[] sqlTypes() { return SQL_TYPES; }
47: public Class returnedClass() { return SiteType.class; }
48: public boolean equals(Object x, Object y) { return x == y; }
49: public Object deepCopy(Object value) { return value; }
50: public boolean isMutable() { return false; }
51:
52: public Object nullSafeGet(ResultSet resultSet,
53:                           String[] names,
54:                           Object owner)
55:     throws HibernateException, SQLException {
56:
57:     String name = resultSet.getString(names[0]);
58:     return resultSet.isNull() ? null : SiteType.getInstance(name);
59: }
60:
61: public void nullSafeSet(PreparedStatement statement,
62:                        Object value,
63:                        int index)
64:     throws HibernateException, SQLException {
65:
66:     if (value == null) {
67:         statement.setNull(index, Types.VARCHAR);
68:     } else {
69:         statement.setString(index, value.toString());
70:     }
71: }
72: public Object assemble(Serializable arg0, Object arg1) throws HibernateException {
73:     // TODO Auto-generated method stub
74:     return null;
75: }
76: public Serializable disassemble(Object arg0) throws HibernateException {
77:     // TODO Auto-generated method stub
78:     return null;
79: }
80: public int hashCode(Object arg0) throws HibernateException {
81:     // TODO Auto-generated method stub
82:     return 0;
83: }
84: public Object replace(Object arg0, Object arg1, Object arg2) throws HibernateException {
85:     // TODO Auto-generated method stub
86:     return null;
87: }
88: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      ActivityEndReasonDescriptor.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: /**
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36:
37: package export.types.descriptors;
38:
39: //-----/
40: //- Imported classes and packages -/
41: //-----/
42:
43: import export.types.ActivityEndReason;
44:
45: /**
```

```
46:  * Class ActivityEndReasonDescriptor.
47:  *
48:  * @version $Revision$ $Date$
49:  */
50: public class ActivityEndReasonDescriptor extends org.exolab.castor.xml.util.XMLClassDescriptorImpl {
51:
52:
53:     //-----/
54:     //- Class/Member Variables -/
55:     //-----/
56:
57:     /**
58:      * Field _elementDefinition.
59:      */
60:     private boolean _elementDefinition;
61:
62:     /**
63:      * Field _nsPrefix.
64:      */
65:     private java.lang.String _nsPrefix;
66:
67:     /**
68:      * Field _nsURI.
69:      */
70:     private java.lang.String _nsURI;
71:
72:     /**
73:      * Field _xmlName.
74:      */
75:     private java.lang.String _xmlName;
76:
77:     /**
78:      * Field _identity.
79:      */
80:     private org.exolab.castor.xml.XMLFieldDescriptor _identity;
81:
82:
83:     //-----/
84:     //- Constructors -/
85:     //-----/
86:
87:     public ActivityEndReasonDescriptor() {
88:         super();
89:         _xmlName = "ActivityEndReason";
90:         _elementDefinition = false;
```

```
91:     }
92:
93:
94:     //-----/
95:     //- Methods -/
96:     //-----/
97:
98:     /**
99:      * Method getAccessMode.
100:      *
101:      * @return the access mode specified for this class.
102:      */
103:     @Override
104:     public org.exolab.castor.mapping.AccessMode getAccessMode(
105:     ) {
106:         return null;
107:     }
108:
109:     /**
110:      * Method getIdentity.
111:      *
112:      * @return the identity field, null if this class has no
113:      * identity.
114:      */
115:     public org.exolab.castor.mapping.FieldDescriptor getIdentity(
116:     ) {
117:         return _identity;
118:     }
119:
120:     /**
121:      * Method getJavaClass.
122:      *
123:      * @return the Java class represented by this descriptor.
124:      */
125:     public java.lang.Class getJavaClass(
126:     ) {
127:         return ActivityEndReason.class;
128:     }
129:
130:     /**
131:      * Method getNameSpacePrefix.
132:      *
133:      * @return the namespace prefix to use when marshaling as XML.
134:      */
135:     public java.lang.String getNameSpacePrefix(
```

```
136:     ) {
137:         return _nsPrefix;
138:     }
139:
140:     /**
141:      * Method getNamespaceURI.
142:      *
143:      * @return the namespace URI used when marshaling and
144:      * unmarshaling as XML.
145:      */
146:     public java.lang.String getNamespaceURI(
147:     ) {
148:         return _nsURI;
149:     }
150:
151:     /**
152:      * Method getValidator.
153:      *
154:      * @return a specific validator for the class described by this
155:      * ClassDescriptor.
156:      */
157:     public org.exolab.castor.xml.TypeValidator getValidator(
158:     ) {
159:         return this;
160:     }
161:
162:     /**
163:      * Method getXMLName.
164:      *
165:      * @return the XML Name for the Class being described.
166:      */
167:     public java.lang.String getXMLName(
168:     ) {
169:         return _xmlName;
170:     }
171:
172:     /**
173:      * Method isElementDefinition.
174:      *
175:      * @return true if XML schema definition of this Class is that
176:      * of a global
177:      * element or element with anonymous type definition.
178:      */
179:     public boolean isElementDefinition(
180:     ) {
```

05/19/09
20:28:58

BIRODatabaseManager/src/export/types/descriptors/ActivityEndReasonDescriptor.java

5

```
181:         return _elementDefinition;  
182:     }  
183:  
184: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      ActivityStartReasonDescriptor.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: /*
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36:
37: package export.types.descriptors;
38:
39:  //---------------------------------/
40:  //- Imported classes and packages -/
41:  //---------------------------------/
42:
43: import export.types.ActivityStartReason;
44:
45: /**
```

```
46:  * Class ActivityStartReasonDescriptor.
47:  *
48:  * @version $Revision$ $Date$
49:  */
50: public class ActivityStartReasonDescriptor extends org.exolab.castor.xml.util.XMLClassDescriptorImpl {
51:
52:
53:     //-----/
54:     //- Class/Member Variables -/
55:     //-----/
56:
57:     /**
58:      * Field _elementDefinition.
59:      */
60:     private boolean _elementDefinition;
61:
62:     /**
63:      * Field _nsPrefix.
64:      */
65:     private java.lang.String _nsPrefix;
66:
67:     /**
68:      * Field _nsURI.
69:      */
70:     private java.lang.String _nsURI;
71:
72:     /**
73:      * Field _xmlName.
74:      */
75:     private java.lang.String _xmlName;
76:
77:     /**
78:      * Field _identity.
79:      */
80:     private org.exolab.castor.xml.XMLFieldDescriptor _identity;
81:
82:
83:     //-----/
84:     //- Constructors -/
85:     //-----/
86:
87:     public ActivityStartReasonDescriptor() {
88:         super();
89:         _xmlName = "ActivityStartReason";
90:         _elementDefinition = false;
```



```
91:     }
92:
93:
94:     //-----/
95:     //- Methods -/
96:     //-----/
97:
98:     /**
99:      * Method getAccessMode.
100:      *
101:      * @return the access mode specified for this class.
102:      */
103:     public org.exolab.castor.mapping.AccessMode getAccessMode(
104:     ) {
105:         return null;
106:     }
107:
108:     /**
109:      * Method getIdentity.
110:      *
111:      * @return the identity field, null if this class has no
112:      * identity.
113:      */
114:     public org.exolab.castor.mapping.FieldDescriptor getIdentity(
115:     ) {
116:         return _identity;
117:     }
118:
119:     /**
120:      * Method getJavaClass.
121:      *
122:      * @return the Java class represented by this descriptor.
123:      */
124:     public java.lang.Class getJavaClass(
125:     ) {
126:         return ActivityStartReason.class;
127:     }
128:
129:     /**
130:      * Method getNameSpacePrefix.
131:      *
132:      * @return the namespace prefix to use when marshaling as XML.
133:      */
134:     public java.lang.String getNameSpacePrefix(
135:     ) {
```

```
136:         return _nsPrefix;
137:     }
138:
139:     /**
140:      * Method getNameSpaceURI.
141:      *
142:      * @return the namespace URI used when marshaling and
143:      * unmarshaling as XML.
144:      */
145:     public java.lang.String getNameSpaceURI(
146:     ) {
147:         return _nsURI;
148:     }
149:
150:     /**
151:      * Method getValidator.
152:      *
153:      * @return a specific validator for the class described by this
154:      * ClassDescriptor.
155:      */
156:     public org.exolab.castor.xml.TypeValidator getValidator(
157:     ) {
158:         return this;
159:     }
160:
161:     /**
162:      * Method getXMLName.
163:      *
164:      * @return the XML Name for the Class being described.
165:      */
166:     public java.lang.String getXMLName(
167:     ) {
168:         return _xmlName;
169:     }
170:
171:     /**
172:      * Method isElementDefinition.
173:      *
174:      * @return true if XML schema definition of this Class is that
175:      * of a global
176:      * element or element with anonymous type definition.
177:      */
178:     public boolean isElementDefinition(
179:     ) {
180:         return _elementDefinition;
```

05/19/09
20:30:38

BIRODatabaseManager/src/export/types/descriptors/ActivityStartReasonDescriptor.java

5

```
181:    }  
182:  
183: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      BIRODatasetDescriptor.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: /**
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36:
37: package export.types.descriptors;
38:
39: /**
40:  * Class BIRODataSetDescriptor.
41:  *
42:  * @version $Revision$ $Date$
43:  */
44: public class BIRODataSetDescriptor extends org.exolab.castor.xml.util.XMLClassDescriptorImpl {
45:
```

```
46:
47:     //-----/
48:     //- Class/Member Variables -/
49:     //-----/
50:
51:     /**
52:      * Field _elementDefinition.
53:      */
54:     private boolean _elementDefinition;
55:
56:     /**
57:      * Field _nsPrefix.
58:      */
59:     private java.lang.String _nsPrefix;
60:
61:     /**
62:      * Field _nsURI.
63:      */
64:     private java.lang.String _nsURI;
65:
66:     /**
67:      * Field _xmlName.
68:      */
69:     private java.lang.String _xmlName;
70:
71:     /**
72:      * Field _identity.
73:      */
74:     private org.exolab.castor.xml.XMLFieldDescriptor _identity;
75:
76:
77:     //-----/
78:     //- Constructors -/
79:     //-----/
80:
81:     public BIRODataSetDescriptor() {
82:         super();
83:         _xmlName = "BIRODataSet";
84:         _elementDefinition = false;
85:     }
86:
87:
88:     //-----/
89:     //- Methods -/
90:     //-----/
```

```
91:
92: /**
93:  * Method getAccessMode.
94:  *
95:  * @return the access mode specified for this class.
96:  */
97: public org.exolab.castor.mapping.AccessMode getAccessMode(
98: ) {
99:     return null;
100: }
101:
102: /**
103:  * Method getIdentity.
104:  *
105:  * @return the identity field, null if this class has no
106:  * identity.
107:  */
108: public org.exolab.castor.mapping.FieldDescriptor getIdentity(
109: ) {
110:     return _identity;
111: }
112:
113: /**
114:  * Method getJavaClass.
115:  *
116:  * @return the Java class represented by this descriptor.
117:  */
118: public java.lang.Class getJavaClass(
119: ) {
120:     return export.types.BIRODataSet.class;
121: }
122:
123: /**
124:  * Method getNameSpacePrefix.
125:  *
126:  * @return the namespace prefix to use when marshaling as XML.
127:  */
128: public java.lang.String getNameSpacePrefix(
129: ) {
130:     return _nsPrefix;
131: }
132:
133: /**
134:  * Method getNameSpaceURI.
135:  *
```

```
136:      * @return the namespace URI used when marshaling and
137:      * unmarshaling as XML.
138:      */
139: public java.lang.String getNamespaceURI(
140: ) {
141:     return _nsURI;
142: }
143:
144: /**
145:  * Method getValidator.
146:  *
147:  * @return a specific validator for the class described by this
148:  * ClassDescriptor.
149:  */
150: public org.exolab.castor.xml.TypeValidator getValidator(
151: ) {
152:     return this;
153: }
154:
155: /**
156:  * Method getXMLName.
157:  *
158:  * @return the XML Name for the Class being described.
159:  */
160: public java.lang.String getXMLName(
161: ) {
162:     return _xmlName;
163: }
164:
165: /**
166:  * Method isElementDefinition.
167:  *
168:  * @return true if XML schema definition of this Class is that
169:  * of a global
170:  * element or element with anonymous type definition.
171:  */
172: public boolean isElementDefinition(
173: ) {
174:     return _elementDefinition;
175: }
176:
177: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      DataSourceDescriptor.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30:
31: /**
32:  * This class was automatically generated with
33:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
34:  * Schema.
35:  * $Id$
36:  */
37:
38: package export.types.descriptors;
39:
40: //-----/
41: //- Imported classes and packages -/
42: //-----/
43:
44: import export.types.DataSource;
45:
```



```
46: /**
47:  * Class DataSourceDescriptor.
48:  *
49:  * @version $Revision$ $Date$
50:  */
51: public class DataSourceDescriptor extends org.exolab.castor.xml.util.XMLClassDescriptorImpl {
52:
53:
54:     //-----/
55:     //- Class/Member Variables -/
56:     //-----/
57:
58:     /**
59:      * Field _elementDefinition.
60:      */
61:     private boolean _elementDefinition;
62:
63:     /**
64:      * Field _nsPrefix.
65:      */
66:     private java.lang.String _nsPrefix;
67:
68:     /**
69:      * Field _nsURI.
70:      */
71:     private java.lang.String _nsURI;
72:
73:     /**
74:      * Field _xmlName.
75:      */
76:     private java.lang.String _xmlName;
77:
78:     /**
79:      * Field _identity.
80:      */
81:     private org.exolab.castor.xml.XMLFieldDescriptor _identity;
82:
83:
84:     //-----/
85:     //- Constructors -/
86:     //-----/
87:
88:     public DataSourceDescriptor() {
89:         super();
90:         _xmlName = "DataSource";
```

```
91:         _elementDefinition = false;
92:     }
93:
94:
95:         //-----/
96:         //- Methods -/
97:         //-----/
98:
99:     /**
100:      * Method getAccessMode.
101:      *
102:      * @return the access mode specified for this class.
103:      */
104:     public org.exolab.castor.mapping.AccessMode getAccessMode(
105:     ) {
106:         return null;
107:     }
108:
109:     /**
110:      * Method getIdentity.
111:      *
112:      * @return the identity field, null if this class has no
113:      * identity.
114:      */
115:     public org.exolab.castor.mapping.FieldDescriptor getIdentity(
116:     ) {
117:         return _identity;
118:     }
119:
120:     /**
121:      * Method getJavaClass.
122:      *
123:      * @return the Java class represented by this descriptor.
124:      */
125:     public java.lang.Class getJavaClass(
126:     ) {
127:         return export.types.DataSource.class;
128:     }
129:
130:     /**
131:      * Method getNamespacePrefix.
132:      *
133:      * @return the namespace prefix to use when marshaling as XML.
134:      */
135:     public java.lang.String getNamespacePrefix(
```

```
136:     ) {
137:         return _nsPrefix;
138:     }
139:
140:     /**
141:      * Method getNamespaceURI.
142:      *
143:      * @return the namespace URI used when marshaling and
144:      * unmarshaling as XML.
145:      */
146:     public java.lang.String getNamespaceURI(
147:     ) {
148:         return _nsURI;
149:     }
150:
151:     /**
152:      * Method getValidator.
153:      *
154:      * @return a specific validator for the class described by this
155:      * ClassDescriptor.
156:      */
157:     public org.exolab.castor.xml.TypeValidator getValidator(
158:     ) {
159:         return this;
160:     }
161:
162:     /**
163:      * Method getXMLName.
164:      *
165:      * @return the XML Name for the Class being described.
166:      */
167:     public java.lang.String getXMLName(
168:     ) {
169:         return _xmlName;
170:     }
171:
172:     /**
173:      * Method isElementDefinition.
174:      *
175:      * @return true if XML schema definition of this Class is that
176:      * of a global
177:      * element or element with anonymous type definition.
178:      */
179:     public boolean isElementDefinition(
180:     ) {
```

05/19/09
20:30:38

BIRODatabaseManager/src/export/types/descriptors/DataSourceDescriptor.java

5

```
181:         return _elementDefinition;  
182:     }  
183:  
184: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      RankingDescriptor.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  **/
30: /*
31:  * This class was automatically generated with
32:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
33:  * Schema.
34:  * $Id$
35:  */
36:
37: package export.types.descriptors;
38:
39:  //-----/
40:  //- Imported classes and packages -/
41:  //-----/
42:
43: import export.types.Ranking;
44:
45: /**
```

```
46:  * Class RankingDescriptor.
47:  *
48:  * @version $Revision$ $Date$
49:  */
50: public class RankingDescriptor extends org.exolab.castor.xml.util.XMLClassDescriptorImpl {
51:
52:
53:     //-----/
54:     //- Class/Member Variables -/
55:     //-----/
56:
57:     /**
58:      * Field _elementDefinition.
59:      */
60:     private boolean _elementDefinition;
61:
62:     /**
63:      * Field _nsPrefix.
64:      */
65:     private java.lang.String _nsPrefix;
66:
67:     /**
68:      * Field _nsURI.
69:      */
70:     private java.lang.String _nsURI;
71:
72:     /**
73:      * Field _xmlName.
74:      */
75:     private java.lang.String _xmlName;
76:
77:     /**
78:      * Field _identity.
79:      */
80:     private org.exolab.castor.xml.XMLFieldDescriptor _identity;
81:
82:
83:     //-----/
84:     //- Constructors -/
85:     //-----/
86:
87:     public RankingDescriptor() {
88:         super();
89:         _xmlName = "Ranking";
90:         _elementDefinition = false;
```

```
91:     }
92:
93:
94:     //-----/
95:     //- Methods -/
96:     //-----/
97:
98:     /**
99:      * Method getAccessMode.
100:     *
101:     * @return the access mode specified for this class.
102:     */
103:     public org.exolab.castor.mapping.AccessMode getAccessMode(
104:     ) {
105:         return null;
106:     }
107:
108:     /**
109:      * Method getIdentity.
110:     *
111:     * @return the identity field, null if this class has no
112:     * identity.
113:     */
114:     public org.exolab.castor.mapping.FieldDescriptor getIdentity(
115:     ) {
116:         return _identity;
117:     }
118:
119:     /**
120:      * Method getJavaClass.
121:     *
122:     * @return the Java class represented by this descriptor.
123:     */
124:     public java.lang.Class getJavaClass(
125:     ) {
126:         return export.types.Ranking.class;
127:     }
128:
129:     /**
130:      * Method getNameSpacePrefix.
131:     *
132:     * @return the namespace prefix to use when marshaling as XML.
133:     */
134:     public java.lang.String getNameSpacePrefix(
135:     ) {
```

```
136:         return _nsPrefix;
137:     }
138:
139:     /**
140:      * Method getNameSpaceURI.
141:      *
142:      * @return the namespace URI used when marshaling and
143:      * unmarshaling as XML.
144:      */
145:     public java.lang.String getNameSpaceURI(
146:     ) {
147:         return _nsURI;
148:     }
149:
150:     /**
151:      * Method getValidator.
152:      *
153:      * @return a specific validator for the class described by this
154:      * ClassDescriptor.
155:      */
156:     public org.exolab.castor.xml.TypeValidator getValidator(
157:     ) {
158:         return this;
159:     }
160:
161:     /**
162:      * Method getXMLName.
163:      *
164:      * @return the XML Name for the Class being described.
165:      */
166:     public java.lang.String getXMLName(
167:     ) {
168:         return _xmlName;
169:     }
170:
171:     /**
172:      * Method isElementDefinition.
173:      *
174:      * @return true if XML schema definition of this Class is that
175:      * of a global
176:      * element or element with anonymous type definition.
177:      */
178:     public boolean isElementDefinition(
179:     ) {
180:         return _elementDefinition;
```


05/19/09
20:30:38

BIRODatabaseManager/src/export/types/descriptors/RankingDescriptor.java

5

```
181:     }  
182:  
183: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      SiteTypeDescriptor.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30:
31: /*
32:  * This class was automatically generated with
33:  * <a href="http://www.castor.org">Castor 1.1</a>, using an XML
34:  * Schema.
35:  * $Id$
36:  */
37:
38: package export.types.descriptors;
39:
40: //-----/
41: //- Imported classes and packages -/
42: //-----/
43:
44: import export.types.SiteType;
45:
```

```
46: /**
47:  * Class SiteTypeDescriptor.
48:  *
49:  * @version $Revision$ $Date$
50:  */
51: public class SiteTypeDescriptor extends org.exolab.castor.xml.util.XMLClassDescriptorImpl {
52:
53:
54:     //-----/
55:     //- Class/Member Variables -/
56:     //-----/
57:
58:     /**
59:      * Field _elementDefinition.
60:      */
61:     private boolean _elementDefinition;
62:
63:     /**
64:      * Field _nsPrefix.
65:      */
66:     private java.lang.String _nsPrefix;
67:
68:     /**
69:      * Field _nsURI.
70:      */
71:     private java.lang.String _nsURI;
72:
73:     /**
74:      * Field _xmlName.
75:      */
76:     private java.lang.String _xmlName;
77:
78:     /**
79:      * Field _identity.
80:      */
81:     private org.exolab.castor.xml.XMLFieldDescriptor _identity;
82:
83:
84:     //-----/
85:     //- Constructors -/
86:     //-----/
87:
88:     public SiteTypeDescriptor() {
89:         super();
90:         _xmlName = "SiteType";
```

```
91:         _elementDefinition = false;
92:     }
93:
94:
95:         //-----/
96:         //- Methods -/
97:         //-----/
98:
99:     /**
100:      * Method getAccessMode.
101:      *
102:      * @return the access mode specified for this class.
103:      */
104:     public org.exolab.castor.mapping.AccessMode getAccessMode(
105:     ) {
106:         return null;
107:     }
108:
109:     /**
110:      * Method getIdentity.
111:      *
112:      * @return the identity field, null if this class has no
113:      * identity.
114:      */
115:     public org.exolab.castor.mapping.FieldDescriptor getIdentity(
116:     ) {
117:         return _identity;
118:     }
119:
120:     /**
121:      * Method getJavaClass.
122:      *
123:      * @return the Java class represented by this descriptor.
124:      */
125:     public java.lang.Class getJavaClass(
126:     ) {
127:         return export.types.SiteType.class;
128:     }
129:
130:     /**
131:      * Method getNameSpacePrefix.
132:      *
133:      * @return the namespace prefix to use when marshaling as XML.
134:      */
135:     public java.lang.String getNameSpacePrefix(
```

```
136:     ) {
137:         return _nsPrefix;
138:     }
139:
140:     /**
141:      * Method getNamespaceURI.
142:      *
143:      * @return the namespace URI used when marshaling and
144:      * unmarshaling as XML.
145:      */
146:     public java.lang.String getNamespaceURI(
147:     ) {
148:         return _nsURI;
149:     }
150:
151:     /**
152:      * Method getValidator.
153:      *
154:      * @return a specific validator for the class described by this
155:      * ClassDescriptor.
156:      */
157:     public org.exolab.castor.xml.TypeValidator getValidator(
158:     ) {
159:         return this;
160:     }
161:
162:     /**
163:      * Method getXMLName.
164:      *
165:      * @return the XML Name for the Class being described.
166:      */
167:     public java.lang.String getXMLName(
168:     ) {
169:         return _xmlName;
170:     }
171:
172:     /**
173:      * Method isElementDefinition.
174:      *
175:      * @return true if XML schema definition of this Class is that
176:      * of a global
177:      * element or element with anonymous type definition.
178:      */
179:     public boolean isElementDefinition(
180:     ) {
```

```
181:         return _elementDefinition;
182:     }
183:
184: }
```

```
1: <?xml version="1.0"?>
2: <!DOCTYPE hibernate-mapping PUBLIC
3:     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4:     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5:
6: <hibernate-mapping>
7:
8:     <class name="export.ActivityData" table="activity_data">
9:         <id name="ID" column="activity_data_id">
10:             <generator class="native"/>
11:         </id>
12:         <property name="hibernateStartDate" type="date" column="start_date"/>
13:         <property name="AD_START_REASON" type="export.types.ActivityStartReasonUserType" column="start_reason"/>
14:         <property name="hibernateEndDate" type="date" column="end_date"/>
15:         <property name="AD_END_REASON" type="export.types.ActivityEndReasonUserType" column="end_reason"/>
16:
17:     </class>
18: </hibernate-mapping>
19:
20:
21:
22:
23:
24:
```

```
1: <?xml version="1.0"?>
2: <!DOCTYPE hibernate-mapping PUBLIC
3:     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4:     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5:
6: <hibernate-mapping>
7:
8:     <class name="export.types.BIRODataSet" table="birodataset">
9:         <id name="type" type="string" column="field_id">
10:         </id>
11:         <property name="stringValue" type="string" column="field_name"/>
12:     </class>
13:
14: </hibernate-mapping>
15:
16:
17:
18:
19:
20:
21:
```



```
1: <?xml version="1.0"?>
2: <!DOCTYPE hibernate-mapping PUBLIC
3:     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4:     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5:
6: <hibernate-mapping>
7:
8:     <class name="export.Data" table="data">
9:         <id name="ID" column="data_id">
10:             <generator class="native"/>
11:         </id>
12:         <property name="EpisodeFieldName" type="export.types.BIRODataSetUserType" column="episode_field_name"/>
13:         <property name="EpisodeFieldValue" type="string" column="episode_field_value"/>
14:
15:
16:     </class>
17: </hibernate-mapping>
18:
19:
20:
21:
22:
23:
```

```
1: <?xml version="1.0"?>
2: <!DOCTYPE hibernate-mapping PUBLIC
3:     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4:     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5:
6: <hibernate-mapping>
7:
8:     <class name="export.DataHeader" table="data_header">
9:         <id name="ID" column="data_header_id">
10:             <generator class="native"/>
11:         </id>
12:         <property name="DateCreated" type="string" column="date_created"/>
13:
14:         <property name="DS_ID" type="export.types.DataSourceUserType" column="ds_id"/>
15:
16:     </class>
17: </hibernate-mapping>
18:
19:
20:
21:
22:
23:
```

```
1: <?xml version="1.0"?>
2: <!DOCTYPE hibernate-mapping PUBLIC
3:     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4:     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5:
6: <hibernate-mapping>
7:
8:     <class name="export.ECDataExport" table="ec_data_export" >
9:         <id name="ID" column="ec_data_export_id">
10:             <generator class="native"/>
11:         </id>
12:         <many-to-one name="Patient"
13:             column="patient_id"
14:             unique="true"
15:             cascade="all"/>
16:         <many-to-one name="DataHeader"
17:             column="data_header_id"
18:             unique="true"
19:             cascade="all"/>
20:     </class>
21: </hibernate-mapping>
22:
```

```
1: <?xml version="1.0"?>
2: <!DOCTYPE hibernate-mapping PUBLIC
3:     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4:     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5:
6: <hibernate-mapping>
7:
8:     <class name="export.ECDataSourceExport" table="ec_data_source_export" >
9:         <id name="ID" column="ec_data_source_export_id">
10:             <generator class="native"/>
11:         </id>
12:         <many-to-one name="siteHeader"
13:             column="site_header_id"
14:             unique="true"
15:             cascade="all"/>
16:         <many-to-one name="SiteProfile"
17:             column="site_profile_id"
18:             unique="true"
19:             cascade="all"/>
20:         <bag name="fieldExportProfilesList" order-by="field_export_profiles_id" cascade="all">
21:             <key column="ec_data_source_export_id" />
22:             <one-to-many class="export.FieldExportProfiles" />
23:         </bag>
24:     </class>
25: </hibernate-mapping>
26:
```

```
1: <?xml version="1.0"?>
2: <!DOCTYPE hibernate-mapping PUBLIC
3:     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4:     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5:
6: <hibernate-mapping>
7:
8:     <class name="export.EpisodeData" table="episode_data" >
9:         <id name="ID" column="episode_data_id">
10:             <generator class="native"/>
11:         </id>
12:         <property name="hibernateEpisodeDate" type="date" column="episode_date"/>
13:         <bag name="DataList" order-by="data_id" cascade="all">
14:             <key column="episode_data_id" />
15:             <one-to-many class="export.Data" />
16:         </bag>
17:     </class>
18: </hibernate-mapping>
19:
```

```
1: <?xml version="1.0"?>
2: <!DOCTYPE hibernate-mapping PUBLIC
3:     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4:     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5:
6: <hibernate-mapping>
7:
8:     <class name="export.Export" table="export" >
9:         <id name="ID" column="export_id">
10:             <generator class="native"/>
11:         </id>
12:         <many-to-one name="ECDataSourceExport"
13:             column="ec_data_source_export_id"
14:             unique="true"
15:             cascade="all"/>
16:         <bag name="ECDataExportList" order-by="ec_data_export_id" cascade="all">
17:             <key column="ec_data_export_id" />
18:             <one-to-many class="export.ECDataExport" />
19:         </bag>
20:
21:     </class>
22: </hibernate-mapping>
23:
```

```
1: <?xml version="1.0"?>
2: <!DOCTYPE hibernate-mapping PUBLIC
3:     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4:     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5:
6: <hibernate-mapping>
7:
8:     <class name="export.FieldExportProfiles" table="field_export_profiles">
9:         <id name="ID" column="field_export_profiles_id">
10:             <generator class="native"/>
11:         </id>
12:         <property name="fieldName" type="export.types.BIRODataSetUserType" column="field_name"/>
13:         <property name="hibernateDateStatusLastReviewed" type="date" column="date_status_last_reviewed"/>
14:         <property name="recorded" type="boolean" column="recorded"/>
15:         <property name="consistency" type="export.types.RankingUserType" column="consistency" not-null="false"/>
16:         <property name="completeness" type="long" column="completeness"/>
17:         <property name="mandatory" type="boolean" column="mandatory"/>
18:         <property name="routine" type="boolean" column="routine"/>
19:         <property name="qualityScore" type="export.types.RankingUserType" column="quality_score"/>
20:         <property name="fieldExportComments" type="string" column="field_export_comments"/>
21:     </class>
22: </hibernate-mapping>
23:
24:
25:
26:
27:
28:
29:
```

```
1: <?xml version="1.0"?>
2: <!DOCTYPE hibernate-mapping PUBLIC
3:     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4:     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5:
6: <hibernate-mapping>
7:
8:     <class name="export.Patient" table="patient" >
9:         <id name="ID" column="patient_id">
10:             <generator class="native"/>
11:         </id>
12:         <bag name="ProfileList" order-by="profile_id" cascade="all">
13:             <key column="patient_id" />
14:             <one-to-many class="export.Profile" />
15:         </bag>
16:         <bag name="ActivityDataList" order-by="activity_data_id" cascade="all">
17:             <key column="patient_id" />
18:             <one-to-many class="export.ActivityData" />
19:         </bag>
20:         <bag name="EpisodeDataList" order-by="episode_data_id" cascade="all">
21:             <key column="patient_id" />
22:             <one-to-many class="export.EpisodeData" />
23:         </bag>
24:     </class>
25: </hibernate-mapping>
26:
```



```
1: <?xml version="1.0"?>
2: <!DOCTYPE hibernate-mapping PUBLIC
3:     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4:     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5:
6: <hibernate-mapping>
7:
8:     <class name="export.Profile" table="profile">
9:         <id name="ID" column="profile_id">
10:             <generator class="native"/>
11:         </id>
12:         <property name="ProfileFieldName" type="export.types.BIRODataSetUserType" column="profile_field_name"/>
13:         <property name="ProfileFieldValue" type="string" column="profile_field_value"/>
14:
15:
16:     </class>
17: </hibernate-mapping>
18:
19:
20:
21:
22:
23:
```

```
1: <?xml version="1.0"?>
2: <!DOCTYPE hibernate-mapping PUBLIC
3:     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4:     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5:
6: <hibernate-mapping>
7:
8:     <class name="export.SiteHeader" table="site_header">
9:         <id name="ID" column="site_header_id">
10:             <generator class="native"/>
11:         </id>
12:         <property name="hibernateDateHeaderInformationChecked" type="date" column="date_header_information_checked"
/>
13:         <property name="DS_ID" type="export.types.DataSourceUserType" column="ds_id"/>
14:         <property name="DS_WEBSITE" type="string" column="ds_website"/>
15:         <property name="DS_ADDRESS_1" type="string" column="ds_address_1"/>
16:         <property name="DS_ADDRESS_2" type="string" column="ds_address_2"/>
17:         <property name="DS_ADDRESS_3" type="string" column="ds_address_3"/>
18:         <property name="DS_ADDRESS_4" type="string" column="ds_address_4"/>
19:         <property name="DS_POST_CODE" type="string" column="ds_post_code"/>
20:         <property name="DS_COUNTRY" type="string" column="ds_country"/>
21:         <property name="DS_C_CONTACT" type="string" column="ds_c_contact"/>
22:         <property name="DS_C_EMAIL" type="string" column="ds_c_email"/>
23:         <property name="DS_T_CONTACT" type="string" column="ds_t_contact"/>
24:         <property name="DS_T_EMAIL" type="string" column="ds_t_email"/>
25:         <property name="headerComments" type="string" column="ds_header_comments"/>
26:     </class>
27: </hibernate-mapping>
28:
29:
30:
31:
32:
33:
```

```
1: <?xml version="1.0"?>
2: <!DOCTYPE hibernate-mapping PUBLIC
3:     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4:     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5:
6: <hibernate-mapping>
7:
8:     <class name="export.SiteProfile" table="site_profile">
9:         <id name="ID" column="site_profile_id">
10:             <generator class="native"/>
11:         </id>
12:         <property name="hibernateDateProfileInformationChecked" type="date" column=
13: "date_profile_information_checked"/>
14:         <property name="DS_TYPE" type="export.types.SiteTypeUserType" column="ds_type"/>
15:         <property name="DS_DENOM" type="long" column="ds_denom"/>
16:         <property name="DS_AREA" type="long" column="ds_area"/>
17:         <property name="DS_BEDS" type="long" column="ds_beds"/>
18:         <property name="DS_PHYSICIANS" type="long" column="ds_physicians"/>
19:         <property name="DS_DIABETOLOGISTS" type="long" column="ds_diabetologists"/>
20:         <property name="DS_DOCTORS" type="long" column="ds_doctors"/>
21:         <property name="DS_DSN" type="long" column="ds_dsn"/>
22:         <property name="DS_DMP_PHYSICIANS" type="long" column="ds_dmp_physicians"/>
23:         <property name="ProfileComments" type="string" column="profile_comments"/>
24:     </class>
25: </hibernate-mapping>
26:
27:
28:
29:
30:
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      MergeCreator.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  * GPL Copyright, The BIRO Project
27:  *
28:  */
29: package mergecreator;
30:
31: import eu.biro.adaptor2.DBMSDriver;
32: import java.sql.Connection;
33: import java.sql.ResultSet;
34: import java.sql.SQLException;
35: import java.sql.Statement;
36: import java.util.Iterator;
37: import java.util.Vector;
38:
39: public class MergeCreator {
40:
41:     private DBMSDriver driver;
42:
43:     public MergeCreator(DBMSDriver driver) {
44:         this.driver = driver;
45:     }
```

```
46:
47: public void createMerge() throws Exception {
48:     Connection connection = null;
49:     try {
50:         this.driver.initialize();
51:         connection = this.driver.openConnection();
52:         createProfileMerge(connection);
53:         createEpisodeMerge(connection);
54:         connection.close();
55:     } catch (Exception e) {
56:         connection.close();
57:         throw e;
58:     }
59: }
60:
61: public void createProfileMerge(Connection connection) throws SQLException {
62:     String name;
63:     String s;
64:     Statement stmt;
65:     boolean first;
66:     Iterator i;
67:
68:     stmt = connection.createStatement();
69:
70:     s = "DROP TABLE IF EXISTS profile_wide";
71:     System.out.println("executing query: " + s);
72:     stmt.executeUpdate(s);
73:
74:     s = "DROP TABLE IF EXISTS joined_table";
75:     System.out.println("executing query: " + s);
76:     stmt.executeUpdate(s);
77:
78:     s = "DROP TABLE IF EXISTS patients";
79:     System.out.println("executing query: " + s);
80:     stmt.executeUpdate(s);
81:
82:     s = "CREATE TABLE patients AS Select DISTINCT patient_id FROM profile";
83:     System.out.println("executing query: " + s);
84:     stmt.executeUpdate(s);
85:
86:     s = "CREATE INDEX patients_index ON patients (patient_id)";
87:     System.out.println("executing query: " + s);
88:     stmt.executeUpdate(s);
89:
90:     s = "SELECT DISTINCT profile_field_name FROM profile";
```

```
91:      ResultSet profileNames = stmt.executeQuery(s);
92:      Vector<String> v = new Vector<String>();
93:
94:      while (profileNames.next()) {
95:          v.add(profileNames.getString(1));
96:      }
97:      //elimino le tabelle singole se esistono
98:      i = v.iterator();
99:      while (i.hasNext()) {
100:          name = (String) i.next();
101:          s = "DROP TABLE IF EXISTS " + name.toLowerCase();
102:          System.out.println("executing query: " + s);
103:          stmt.executeUpdate(s);
104:      }
105:
106:
107:      //creo le tabelle singole
108:      i = v.iterator();
109:
110:      while (i.hasNext()) {
111:          name = (String) i.next();
112:          s = "CREATE TABLE " + name.toLowerCase() + " AS SELECT DISTINCT profile.profile_field_value AS " +
name.toLowerCase() + ", profile.patient_id FROM profile WHERE profile_field_name='" + name + "'";
113:          System.out.println("executing query: " + s);
114:          stmt.executeUpdate(s);
115:      }
116:
117:      //creo gli indici delle tabelle singole
118:      i = v.iterator();
119:
120:      while (i.hasNext()) {
121:          name = (String) i.next();
122:          s = "CREATE INDEX " + name.toLowerCase() + "_index ON " + name.toLowerCase() + " (patient_id)";
123:          System.out.println("executing query: " + s);
124:          stmt.executeUpdate(s);
125:      }
126:
127:
128:
129:      first = true;
130:      //creo la tabella di join
131:      i = v.iterator();
132:      while (i.hasNext()) {
133:          name = (String) i.next();
134:          if (first) {
```

```
135:         s = "CREATE TABLE joined_table AS SELECT patients.*, " + name.toLowerCase() + "." +
name.toLowerCase() + " FROM patients LEFT JOIN " + name.toLowerCase() + " ON " + name.toLowerCase() + ".patient_id =
patients.patient_id ";
136:         System.out.println("executing query: " + s);
137:         stmt.executeUpdate(s);
138:         first = false;
139:     } else {
140:         s = "CREATE TABLE joined_table_2 AS SELECT joined_table.*, " + name.toLowerCase() + "." +
name.toLowerCase() + " FROM joined_table LEFT JOIN " + name.toLowerCase() + " ON " + name.toLowerCase() + ".patient_id =
joined_table.patient_id ";
141:         System.out.println("executing query: " + s);
142:         stmt.executeUpdate(s);
143:         s = "DROP TABLE IF EXISTS joined_table";
144:         System.out.println("executing query: " + s);
145:         stmt.executeUpdate(s);
146:         s = "ALTER TABLE joined_table_2 RENAME TO joined_table";
147:         System.out.println("executing query: " + s);
148:         stmt.executeUpdate(s);
149:
150:     }
151: }
152:
153: s = "ALTER TABLE joined_table RENAME TO profile_wide";
154: System.out.println("executing query: " + s);
155: stmt.executeUpdate(s);
156:
157: //cancello tutte le tabelle singole
158: i = v.iterator();
159: while (i.hasNext()) {
160:     name = (String) i.next();
161:     s = "DROP TABLE IF EXISTS " + name.toLowerCase();
162:     System.out.println("executing query: " + s);
163:     stmt.executeUpdate(s);
164: }
165:
166: //cancello la tabella patients
167: s = "DROP TABLE IF EXISTS patients";
168: System.out.println("executing query: " + s);
169: stmt.executeUpdate(s);
170: }
171:
172: public void createEpisodeMerge(Connection connection) throws SQLException {
173:     String name;
174:     String s;
175:     Statement stmt;
```

```
176:      ResultSet episodeNames;
177:      Vector<String> v;
178:      Iterator i;
179:      boolean first;
180:
181:      stmt = connection.createStatement();
182:
183:      s = "DROP TABLE IF EXISTS episode_wide";
184:      System.out.println("executing query: " + s);
185:      stmt.executeUpdate(s);
186:
187:      s = "DROP TABLE IF EXISTS joined_table";
188:      System.out.println("executing query: " + s);
189:      stmt.executeUpdate(s);
190:
191:
192:      s = "SELECT DISTINCT episode_field_name FROM data";
193:      episodeNames = stmt.executeQuery(s);
194:      v = new Vector<String>();
195:
196:      while (episodeNames.next()) {
197:          v.add(episodeNames.getString(1));
198:      }
199:
200:
201:      //elimino le tabelle singole se esistono
202:      i = v.iterator();
203:      while (i.hasNext()) {
204:          name = (String) i.next();
205:          s = "DROP TABLE IF EXISTS " + name.toLowerCase();
206:          System.out.println("executing query: " + s);
207:          stmt.executeUpdate(s);
208:      }
209:
210:
211:      //creo le tabelle singole
212:      i = v.iterator();
213:
214:      while (i.hasNext()) {
215:          name = (String) i.next();
216:          s = "CREATE TABLE " + name.toLowerCase() + " AS SELECT DISTINCT data.episode_field_value AS " +
name.toLowerCase() + ", data.episode_data_id FROM data WHERE episode_field_name='" + name + "'";
217:          System.out.println("executing query: " + s);
218:          stmt.executeUpdate(s);
219:      }
```



```
220:
221:         //creo gli indici delle tabelle singole
222:         i = v.iterator();
223:
224:         while (i.hasNext()) {
225:             name = (String) i.next();
226:             s = "CREATE INDEX " + name.toLowerCase() + "_index ON " + name.toLowerCase() + " (episode_data_id)";
227:             System.out.println("executing query: " + s);
228:             stmt.executeUpdate(s);
229:         }
230:
231:         first = true;
232:         //creo la tabella di join
233:         i = v.iterator();
234:         while (i.hasNext()) {
235:             name = (String) i.next();
236:             if (first) {
237:                 s = "CREATE TABLE joined_table AS SELECT episode_data.episode_data_id, episode_data.patient_id," +
name.toLowerCase() + "." + name.toLowerCase() + " FROM episode_data LEFT JOIN " + name.toLowerCase() + " ON " +
name.toLowerCase() + ".episode_data_id = episode_data.episode_data_id ";
238:                 System.out.println("executing query: " + s);
239:                 stmt.executeUpdate(s);
240:                 first = false;
241:             } else {
242:                 s = "CREATE TABLE joined_table_2 AS SELECT joined_table.episode_data_id, joined_table.patient_id," +
+ name.toLowerCase() + "." + name.toLowerCase() + " FROM joined_table LEFT JOIN " + name.toLowerCase() + " ON " +
name.toLowerCase() + ".episode_data_id = joined_table.episode_data_id ";
243:                 System.out.println("executing query: " + s);
244:                 stmt.executeUpdate(s);
245:                 s = "DROP TABLE IF EXISTS joined_table";
246:                 System.out.println("executing query: " + s);
247:                 stmt.executeUpdate(s);
248:                 s = "ALTER TABLE joined_table_2 RENAME TO joined_table";
249:                 System.out.println("executing query: " + s);
250:                 stmt.executeUpdate(s);
251:             }
252:         }
253:
254:         s = "ALTER TABLE joined_table RENAME TO episode_wide";
255:         System.out.println("executing query: " + s);
256:         stmt.executeUpdate(s);
257:
258:         //cancello tutte le tabelle singole
259:         i = v.iterator();
260:         while (i.hasNext()) {
```

```
261:         name = (String) i.next();
262:         s = "DROP TABLE IF EXISTS " + name.toLowerCase();
263:         System.out.println("executing query: " + s);
264:         stmt.executeUpdate(s);
265:     }
266:
267:     ResultSet rs = connection.getMetaData().getColumns(null, null, "episode_wide", "epi_date");
268:
269:     if (!rs.next())
270:     {
271:         s = "CREATE INDEX episode_wide_index ON episode_wide(episode_data_id)";
272:         System.out.println("executing query: " + s);
273:         stmt.executeUpdate(s);
274:
275:         s = "CREATE INDEX episode_data_index ON episode_data(episode_data_id)";
276:         System.out.println("executing query: " + s);
277:         stmt.executeUpdate(s);
278:
279:         s = "CREATE TABLE episode_wide_2 AS SELECT episode_wide.*,episode_data.episode_date AS epi_date FROM
episode_wide JOIN episode_data ON episode_data.episode_data_id = episode_wide.episode_data_id";
280:         System.out.println("executing query: " + s);
281:         stmt.executeUpdate(s);
282:
283:         s = "DROP TABLE episode_wide";
284:         System.out.println("executing query: " + s);
285:         stmt.executeUpdate(s);
286:
287:         s = "ALTER TABLE episode_wide_2 RENAME TO episode_wide";
288:         System.out.println("executing query: " + s);
289:         stmt.executeUpdate(s);
290:
291:     }
292:
293:
294:     connection.close();
295:
296: }
297: }
298:
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      BIROClassGenerator.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30:
31: package test;
32:
33: import org.exolab.castor.builder.SourceGenerator;
34:
35: /**
36:  * This simple class allow to read the BIRO XML schema and
37:  * to automatically generate Java classes.
38:  *
39:  */
40: public class BIROClassGenerator {
41:
42:     /**
43:      * Execute BIROClassGenerator
44:      * It requires as arguments:
45:      * 0) Input File Name of the binding file
```

BIRODatabaseManager/src/test/BIROClassGenerator.java

```
46:      * 1) Input File Name of the BIRO XML schema
47:      * 2) Output Directory where Java classes will be stored
48:      */
49:      public static void main(String[] args) {
50:          try {
51:
52:              if (args.length < 3) {
53:                  System.err.println("\nUsage:\n\t" + BIROClassGenerator.class.getName() + " [bindingFile] [XMLschema]
[outputDirectory]");
54:                  return;
55:              }
56:
57:              //istanziare the source genarator
58:              SourceGenerator sourceGen = new SourceGenerator();
59:              String bindingFile = args[0];
60:              String XMLschema = args[1];
61:              String outputDirectory = args[2];
62:
63:              //set the binding file
64:
65:              sourceGen.setBinding(bindingFile);
66:
67:              //generate the object-model from the XML schema
68:              sourceGen.generateSource(XMLschema, outputDirectory);
69:
70:          } catch (Exception e) {
71:              System.out.println(e.toString());
72:          }
73:
74:      }
75: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      BIRODatabaseManager.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: package test;
31:
32: import export.ECDataExport;
33: import export.ECDataSourceExport;
34: import java.io.File;
35: import java.io.FilenameFilter;
36: import java.io.IOException;
37: import java.io.InputStream;
38: import java.io.InputStreamReader;
39: import java.io.Reader;
40: import java.util.Enumeration;
41: import java.util.zip.ZipEntry;
42: import java.util.zip.ZipFile;
43: import javax.swing.event.EventListenerList;
44: import org.exolab.castor.mapping.MappingException;
45: import org.exolab.castor.xml.MarshalException;
```

```
46: import org.exolab.castor.xml.ValidationException;
47: import util.HibernateUtil;
48: import util.DatabaseManagerProgressListener;
49:
50: /**
51:  *
52:  * @author Valentina
53:  */
54: public class BIRODatabaseManager {
55:
56:     private final transient EventListenerList listeners;
57:     private String directory;
58:     private String driverClass;
59:     private String dialect;
60:     private String url;
61:     private String username;
62:     private String password;
63:
64:     public BIRODatabaseManager(String directory, String driverClass, String dialect, String url, String username,
String password) {
65:         this.listeners = new EventListenerList();
66:         this.dialect = dialect;
67:         this.directory = directory;
68:         this.driverClass = driverClass;
69:         this.password = password;
70:         this.url = url;
71:         this.username = username;
72:     }
73:
74:     public synchronized void addProgressListener(DatabaseManagerProgressListener l) {
75:         listeners.add(DatabaseManagerProgressListener.class, l);
76:     }
77:
78:     public synchronized void removeProgressListener(DatabaseManagerProgressListener l) {
79:         listeners.remove(DatabaseManagerProgressListener.class, l);
80:     }
81:
82:     protected void fireSetProgressMaximum(int m) {
83:         for (DatabaseManagerProgressListener l : listeners.getListeners(DatabaseManagerProgressListener.class)) {
84:             l.setMaximumForProgress(m);
85:         }
86:     }
87:
88:     protected void fireShowText(String text) {
89:         for (DatabaseManagerProgressListener l : listeners.getListeners(DatabaseManagerProgressListener.class)) {
```

```
90:         l.showText(text);
91:     }
92: }
93:
94: protected void fireUpdateProgressValue() {
95:     for (DatabaseManagerProgressListener l : listeners.getListeners(DatabaseManagerProgressListener.class)) {
96:         l.updateProgressValue();
97:     }
98: }
99:
100: class ECDataExportFilter implements FilenameFilter {
101:
102:     public boolean accept(File dir, String name) {
103:         return ((name.substring(0, name.length() - 4)).matches("[0-9]*") && name.endsWith(".xml"));
104:     }
105:
106:     private boolean accept(String name) {
107:         return ((name.substring(0, name.length() - 4)).matches("[0-9]*") && name.endsWith(".xml"));
108:     }
109: }
110:
111: class ECDataSourceExportFilter implements FilenameFilter {
112:
113:     public boolean accept(File dir, String name) {
114:         return (name.startsWith("DataSource") && name.endsWith(".xml"));
115:     }
116:
117:     private boolean accept(String name) {
118:         return (name.startsWith("DataSource") && name.endsWith(".xml"));
119:     }
120: }
121:
122: public void unmarshallAndStoreDirectory() throws IOException, MappingException, MarshalException,
ValidationException {
123:
124:     HibernateUtil.setSessionFactory(driverClass, dialect, url, username, password);
125:     ECDataExportFilter eCDataExportFilter = new ECDataExportFilter();
126:     ECDataSourceExportFilter eCDataSourceExportFilter = new ECDataSourceExportFilter();
127:     ZipFile zip = new ZipFile(directory);
128:     Enumeration<? extends ZipEntry> en = zip.entries();
129:     fireSetProgressMaximum(zip.size());
130:     fireShowText("Reading " + zip.size() + " files [may take a while]");
131:     boolean succesful = false;
132:     while (en.hasMoreElements()) {
133:         ZipEntry e = en.nextElement();
```

```
134: //      System.out.println("Reading " + e);
135:      InputStream in = zip.getInputStream(e);
136:      Reader r = new InputStreamReader(in);
137:      try {
138:          if (eCDataExportFilter.accept(e.getName())) {
139:
140:              succesful = UnmarshallingAndStoringManager.unmarshallAndStore(ECDataExport.class, r);
141:              if (!succesful) {
142:                  throw new Exception();
143:              }
144:          } else if (eCDataSourceExportFilter.accept(e.getName())) {
145:
146:              succesful = UnmarshallingAndStoringManager.unmarshallAndStore(ECDataSourceExport.class, r);
147:              if (!succesful) {
148:                  throw new Exception();
149:              }
150:          } else {
151:              throw new IllegalArgumentException("Not valid zip entry " + e);
152:          }
153:          System.gc();
154:          fireUpdateProgressValue();
155:      } catch (Exception ex) {
156:          fireShowText("Failed persistence of file " + e.getName() + ": " + ex.getMessage());
157:      }
158:  }
159:  zip.close();
160:
161:  }
162: }
163:
164:
```



```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      BIRODatabaseManagerMain.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: package test;
31:
32:
33: import util.Loader;
34:
35: /**
36:  * This simple class allow one partner to read XML files following the BIRO XML schema,
37:  * and store data into the local BIRO Database.(see documentation)
38:  *
39:  * @author Valentina Baglioni
40:  */
41: public final class BIRODatabaseManagerMain {
42:
43:
44:
45:     /**
```

```
46:      * Execute BIRO Database Manager
47:      * It requires as arguments:
48:      * 0) Input File Name of the XML Configuration
49:      * 1) Input Directory of the BIRO Xml files
50:      */
51:  public static void main(String[] args) throws Exception {
52:
53:      if (args.length < 6) {
54:          System.err.println("\nUsage:\n\t" + BIRODatabaseManagerMain.class.getName() + "
[InputDir][driverClass][dialect][url][username][password]");
55:          return;
56:      }
57:
58:      Loader.addDir("lib");
59:
60:      String input = args[0];
61:      String driverClass = args[1];
62:      String dialect = args[2];
63:      String url = args[3];
64:      String username = args[4];
65:      String password = args[5];
66:
67:      BIRODatabaseManager biRODatabaseManager = new BIRODatabaseManager(input, driverClass, dialect, url,
username, password);
68:      biRODatabaseManager.unmarhallAndStoreDirectory();
69:  }
70: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      UnmarshallingAndStoringManager.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: package test;
31:
32: import java.io.FileReader;
33: import java.io.IOException;
34:
35: import java.io.Reader;
36: import java.sql.SQLException;
37: import org.exolab.castor.mapping.MappingException;
38: import org.exolab.castor.xml.MarshalException;
39: import org.exolab.castor.xml.Unmarshaller;
40: import org.exolab.castor.xml.ValidationException;
41: import org.hibernate.Session;
42:
43: import org.hibernate.Transaction;
44: import util.HibernateUtil;
45:
```

```
46: /**
47:  * This simple class allow to unmarshall and store a BIRO Export XML file
48:  *
49:  * @author Valentina Baglioni
50:  */
51: public class UnmarshallingAndStoringManager {
52:
53:     public static void unmarshallAndStore(Class<?> c, String f) throws Exception{
54:         FileReader r = new FileReader(f);
55:         unmarshallAndStore(c, r);
56:         r.close();
57:     }
58:
59:     public static boolean unmarshallAndStore(Class<?> c, Reader r) throws Exception {
60:         Object o = unmarshall(c, r);
61:         boolean succesfulStore = store(o);
62:         return succesfulStore;
63:     }
64:
65:     private static Object unmarshall(Class<?> c, Reader reader) throws Exception {
66:         return Unmarshaller.unmarshal(c, reader);
67:     }
68:
69:     private static boolean store(Object o) throws Exception {
70:         boolean succesful = true;
71:         Session session = HibernateUtil.getSessionFactory().getCurrentSession();
72:         Transaction transaction = session.beginTransaction();
73:
74:         try {
75:             session.persist(o);
76:             transaction.commit();
77:             HibernateUtil.getSessionFactory().close();
78:         } catch (Exception e) {
79:             if (!transaction.wasRolledBack() && !transaction.wasCommitted()) {
80:                 transaction.rollback();
81:                 transaction.commit();
82:             }
83:
84:             throw new Exception (e.getMessage());
85:         }
86:         return succesful;
87:     }
88: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      DatabaseManagerProgressListener.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * don't charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: package util;
31:
32: import java.util.EventListener;
33:
34:
35: public interface DatabaseManagerProgressListener
36:     extends EventListener {
37:
38:     void setMaximumForProgress(int max);
39:
40:     void showText(String txt);
41:
42:     void updateProgressValue();
43: }
```

```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      HibbrnateUtil.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * donât charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30: package util;
31:
32: import java.io.File;
33: import java.io.FileInputStream;
34: import java.util.Properties;
35:
36: import org.hibernate.SessionFactory;
37: import org.hibernate.cfg.Configuration;
38:
39:
40: /**
41:  * This simple class allow Hibernate to read the Configuration File where
42:  * database connection settings are stored
43:  *
44:  * @author Valentina Baglioni
45:  */
```

```
46: public class HibernateUtil {
47:
48:     private static SessionFactory sessionFactory;
49:     private static File configurationFile;
50:
51:     /**
52:      * Set a session factory
53:      * @param configFile
54:      *      File representing the configuration file
55:      *
56:      */
57:     public static void setSessionFactory(String driverClass, String dialect, String url, String username, String
password) {
58:         try {
59:
60:             Configuration cfg = new Configuration();
61:
62:             // Database connection settings are stored in the configuration file
63:
64:             cfg.setProperty("hibernate.connection.driver_class", driverClass);
65:             cfg.setProperty("hibernate.dialect", dialect);
66:             cfg.setProperty("hibernate.connection.url", url);
67:             cfg.setProperty("hibernate.connection.username", username);
68:             cfg.setProperty("hibernate.connection.password", password);
69:
70:             // JDBC connection pool (use the built-in)
71:             cfg.setProperty("hibernate.connection.pool_size", "1");
72:             // Enable Hibernate's automatic session context management
73:             cfg.setProperty("hibernate.current_session_context_class", "thread");
74:             // Disable the second-level cache
75:             cfg.setProperty("hibernate.cache.provider_class", "org.hibernate.cache.NoCacheProvider");
76:             // Echo all executed SQL to stdout
77:             cfg.setProperty("hibernate.show_sql", "false");
78:             // Drop and re-create the database schema on startup
79:             cfg.setProperty("hibernate.hbm2ddl.auto", "create");
80:
81:             cfg.addResource("hibernateMappings/ECDataSourceExport.hbm.xml");
82:             cfg.addResource("hibernateMappings/SiteHeader.hbm.xml");
83:             cfg.addResource("hibernateMappings/SiteProfile.hbm.xml");
84:             cfg.addResource("hibernateMappings/FieldExportProfiles.hbm.xml");
85:             cfg.addResource("hibernateMappings/BIRODataSet.hbm.xml");
86:             cfg.addResource("hibernateMappings/ECDataExport.hbm.xml");
87:             cfg.addResource("hibernateMappings/Patient.hbm.xml");
88:             cfg.addResource("hibernateMappings/Profile.hbm.xml");
89:             cfg.addResource("hibernateMappings/EpisodeData.hbm.xml");
```

```
90:         cfg.addResource("hibernateMappings/Data.hbm.xml");
91:         //cfg.addResource("hibernateMappings/Export.hbm.xml");
92:         cfg.addResource("hibernateMappings/ActivityData.hbm.xml");
93:         cfg.addResource("hibernateMappings/DataHeader.hbm.xml");
94:
95:         sessionFactory = cfg.buildSessionFactory();
96:
97:
98:     } catch (Throwable ex) {
99:         System.err.println("Initial SessionFactory creation failed." + ex);
100:        throw new ExceptionInInitializerError(ex);
101:    }
102:
103: }
104:
105: /**
106:  * return the session factory
107:  */
108: public static SessionFactory getSessionFactory() {
109:     return sessionFactory;
110: }
111: }
112:
```



```
1: /**
2:  * Project: BIRO-Project (Funded by European Commission 2005-2008)
3:  *
4:  * File:      Loader.java
5:  * Authors:   Pietro Palladino, Valentina Baglioni
6:  *
7:  * COPYRIGHT INFORMATION
8:  * This file is free software; you can redistribute it and/or modify
9:  * it under the terms of the GNU General Public License as published by
10:  * the Free Software Foundation; either version 2, or (at your option)
11:  * any later version.
12:  *
13:  * This file is distributed in the hope that it will be useful,
14:  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15:  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16:  * GNU General Public License for more details.
17:  *
18:  * You should have received a copy of the GNU General Public License
19:  * along with this file; see the file COPYING. If not, write to
20:  * the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.
21:  *
22:  * In short: you may use this file any way you like, as long as you
23:  * donâ200\231t charge money for it, remove this notice, or hold anyone liable
24:  * for its results.
25:  *
26:  *
27:  * GPL Copyright, The BIRO Project
28:  *
29:  */
30:
31: package util;
32:
33: import java.io.File;
34: import java.io.FilenameFilter;
35: import java.io.IOException;
36: import java.lang.reflect.Method;
37: import java.net.URL;
38: import java.net.URLClassLoader;
39:
40: /**
41:  * Utility class for jar resources.
42:  */
43: public final class Loader {
44:
45:     private Loader() {
```

```
46:     }
47:
48:     /**
49:      * Add a dir to classpath loading Jar and Natives
50:      *
51:      * @param dir
52:      */
53:     public static void addDir(String dir) {
54:         if (dir == null)
55:             return;
56:
57:         final class JarFilter implements FilenameFilter {
58:
59:             public boolean accept(File dir, String name) {
60:                 return name.endsWith(".jar");
61:             }
62:         }
63:
64:         File d = new File(dir);
65:         if (!d.exists())
66:             return;
67:
68:         FilenameFilter jar = new JarFilter();
69:
70:         for (File f : d.listFiles(jar))
71:             try {
72:                 loadJarFile(f);
73:             } catch (IOException e) {
74:             }
75:     }
76:
77:     /**
78:      * Add jar to classpath when out of classpath.
79:      */
80:     private static void addURL(URL u) throws IOException {
81:         URLClassLoader sysloader = (URLClassLoader) ClassLoader.getSystemClassLoader();
82:         Class<?> sysclass = URLClassLoader.class;
83:         try {
84:             Method method = sysclass.getDeclaredMethod("addURL", URL.class);
85:             method.setAccessible(true);
86:             method.invoke(sysloader, u);
87:         } catch (Throwable t) {
88:             throw new IOException("Error, could not add URL to system classloader:\n"
89:                 + t.getMessage());
90:         }
```

```
91:     }
92:
93:     /**
94:      * Add jar to classpath when out of classpath.
95:      */
96:     public static void loadJarFile(File f) throws IOException {
97:         addURL(f.toURI().toURL());
98:     }
99:
100:    /**
101:     * Load native library.
102:     */
103:    public static void loadNativeLib(String path, String name) {
104:        String full_name = (path == null ? "." : path) + "/" + name + ".dll";
105:        System.load(new File(full_name).getAbsolutePath());
106:    }
107: }
```

Table of Contents

1	BIRODatabaseManager/src/export/ActivityData.java	7 pages	303 lines	09/06/22 23:08:56
2	BIRODatabaseManager/src/export/DataHeader.java	6 pages	257 lines	09/05/19 20:20:38
3	BIRODatabaseManager/src/export/Data.java	6 pages	230 lines	09/05/19 20:20:28
4	BIRODatabaseManager/src/export/ECDataExport.java	5 pages	213 lines	09/05/19 20:22:36
5	BIRODatabaseManager/src/export/ECDataSourceExport.java	10 pages	417 lines	09/05/19 20:22:36
6	BIRODatabaseManager/src/export/EpisodeData.java	10 pages	413 lines	09/05/19 20:22:36
7	BIRODatabaseManager/src/export/Export.java	9 pages	382 lines	09/05/19 20:22:36
8	BIRODatabaseManager/src/export/FieldExportProfiles.java	13 pages	584 lines	09/05/19 20:22:36
9	BIRODatabaseManager/src/export/Patient.java	15 pages	667 lines	09/05/19 20:22:36
10	BIRODatabaseManager/src/export/Profile.java	6 pages	230 lines	09/05/19 20:22:36
11	BIRODatabaseManager/src/export/SiteHeader.java	13 pages	569 lines	09/05/19 20:22:36
12	BIRODatabaseManager/src/export/SiteProfile.java	16 pages	701 lines	09/05/19 20:22:36
13	BIRODatabaseManager/src/export/descriptors/ActivityDataDescriptor.java	8 pages	339 lines	09/06/22 23:12:28
14	BIRODatabaseManager/src/export/descriptors/DataDescriptor.java	6 pages	268 lines	09/05/19 20:23:56
15	BIRODatabaseManager/src/export/descriptors/DataHeaderDescriptor.java	6 pages	259 lines	09/05/19 20:23:56
16	BIRODatabaseManager/src/export/descriptors/ECDataExportDescriptor.java	6 pages	252 lines	09/05/19 20:23:56
17	BIRODatabaseManager/src/export/descriptors/ECDataSourceExportDescriptor.java	7 pages	304 lines	09/05/19 20:23:56
18	BIRODatabaseManager/src/export/descriptors/EpisodeDataDescriptor.java	7 pages	272 lines	09/05/19 20:24:08
19	BIRODatabaseManager/src/export/descriptors/FieldExportProfilesDescriptor.java	13 pages	571 lines	09/05/19 20:25:28
20	BIRODatabaseManager/src/export/descriptors/PatientDescriptor.java	8 pages	313 lines	09/05/19 20:25:28
21	BIRODatabaseManager/src/export/descriptors/ProfileDescriptor.java	7 pages	270 lines	09/05/19 20:25:26
22	BIRODatabaseManager/src/export/descriptors/SiteHeaderDescriptor.java	18 pages	762 lines	09/05/19 20:25:26
23	BIRODatabaseManager/src/export/descriptors/SiteProfileDescriptor.java	15 pages	641 lines	09/05/19 20:25:26
24	BIRODatabaseManager/src/export/types/ActivityEndReason.java	5 pages	217 lines	09/05/19 20:28:12
25	BIRODatabaseManager/src/export/types/ActivityEndReasonUserType.java	3 pages	98 lines	09/05/19 20:28:12
26	BIRODatabaseManager/src/export/types/ActivityStartReason.java	5 pages	216 lines	09/05/19 20:28:12
27	BIRODatabaseManager/src/export/types/ActivityStartReasonUserType.java	3 pages	99 lines	09/05/19 20:28:10
28	BIRODatabaseManager/src/export/types/BIRODataSet.java	19 pages	811 lines	09/06/23 15:26:30
29	BIRODatabaseManager/src/export/types/BIRODataSetUserType.java	2 pages	88 lines	09/05/19 20:28:12
30	BIRODatabaseManager/src/export/types/DataSource.java	7 pages	286 lines	09/05/19 20:28:12
31	BIRODatabaseManager/src/export/types/DataSourceUserType.java	2 pages	88 lines	09/05/19 20:28:12
32	BIRODatabaseManager/src/export/types/Ranking.java	6 pages	244 lines	09/05/19 20:28:12
33	BIRODatabaseManager/src/export/types/RankingUserType.java	2 pages	88 lines	09/05/19 20:28:12
34	BIRODatabaseManager/src/export/types/SiteType.java	8 pages	359 lines	09/05/19 20:28:12
35	BIRODatabaseManager/src/export/types/SiteTypeUserType.java	2 pages	88 lines	09/05/19 20:28:10
36	BIRODatabaseManager/src/export/types/descriptors/ActivityEndReasonDescriptor.java	5 pages	184 lines	09/05/19 20:28:58
37	BIRODatabaseManager/src/export/types/descriptors/ActivityStartReasonDescriptor.java	5 pages	183 lines	09/05/19 20:30:38
38	BIRODatabaseManager/src/export/types/descriptors/BIRODataSetDescriptor.java	4 pages	177 lines	09/05/19 20:30:38
39	BIRODatabaseManager/src/export/types/descriptors/DataSourceDescriptor.java	5 pages	184 lines	09/05/19 20:30:38
40	BIRODatabaseManager/src/export/types/descriptors/RankingDescriptor.java	5 pages	183 lines	09/05/19 20:30:38
41	BIRODatabaseManager/src/export/types/descriptors/SiteTypeDescriptor.java	5 pages	184 lines	09/05/19 20:30:38
42	BIRODatabaseManager/src/hibernateMappings/ActivityData.hbm.xml	1 pages	24 lines	09/06/23 15:20:52
43	BIRODatabaseManager/src/hibernateMappings/BIRODataSet.hbm.xml	1 pages	21 lines	09/06/20 15:35:20

Table of Contents

44	BIRODatabaseManager/src/hibernateMappings/Data.hbm.xml	1 pages	23 lines	09/06/20 15:35:20
45	BIRODatabaseManager/src/hibernateMappings/DataHeader.hbm.xml	1 pages	23 lines	09/06/20 15:36:08
46	BIRODatabaseManager/src/hibernateMappings/ECDataExport.hbm.xml	1 pages	22 lines	09/06/20 15:39:06
47	BIRODatabaseManager/src/hibernateMappings/ECDataSourceExport.hbm.xml	1 pages	26 lines	09/06/20 15:39:06
48	BIRODatabaseManager/src/hibernateMappings/EpisodeData.hbm.xml	1 pages	19 lines	09/06/20 15:53:06
49	BIRODatabaseManager/src/hibernateMappings/Export.hbm.xml	1 pages	23 lines	09/06/20 15:54:22
50	BIRODatabaseManager/src/hibernateMappings/FieldExportProfiles.hbm.xml	1 pages	29 lines	09/06/20 15:56:36
51	BIRODatabaseManager/src/hibernateMappings/Patient.hbm.xml	1 pages	26 lines	09/06/20 15:58:14
52	BIRODatabaseManager/src/hibernateMappings/Profile.hbm.xml	1 pages	23 lines	09/06/20 15:59:14
53	BIRODatabaseManager/src/hibernateMappings/SiteHeader.hbm.xml	1 pages	33 lines	09/06/20 16:02:22
54	BIRODatabaseManager/src/hibernateMappings/SiteProfile.hbm.xml	1 pages	30 lines	09/06/20 16:04:56
55	BIRODatabaseManager/src/mergecreator/MergeCreator.java	7 pages	298 lines	09/07/12 12:15:36
56	BIRODatabaseManager/src/test/BIROClassGenerator.java	2 pages	75 lines	09/07/15 13:24:26
57	BIRODatabaseManager/src/test/BIRODatabaseManager.java	4 pages	163 lines	09/07/15 13:24:52
58	BIRODatabaseManager/src/test/BIRODatabaseManagerMain.java	2 pages	70 lines	09/05/19 20:36:04
59	BIRODatabaseManager/src/test/UnmarshallingAndStoringManager.java	2 pages	87 lines	09/07/15 13:25:24
60	BIRODatabaseManager/src/util/DatabaseManagerProgressListener.java	1 pages	43 lines	09/05/19 20:36:04
61	BIRODatabaseManager/src/util/HibernateUtil.java	3 pages	111 lines	09/05/19 20:36:04
62	BIRODatabaseManager/src/util/Loader.java	3 pages	107 lines	09/05/19 20:36:04