001/\*  
002 \* Licensed to the Apache Software Foundation (ASF) under one or more  
003 \* contributor license agreements. See the NOTICE file distributed with  
004 \* this work for additional information regarding copyright ownership.  
005 \* The ASF licenses this file to You under the Apache License, Version 2.0  
006 \* (the "License"); you may not use this file except in compliance with  
007 \* the License. You may obtain a copy of the License at  
008 \*  
009 \* http://www.apache.org/licenses/LICENSE-2.0  
010 \*  
011 \* Unless required by applicable law or agreed to in writing, software  
012 \* distributed under the License is distributed on an "AS IS" BASIS,  
013 \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
014 \* See the License for the specific language governing permissions and  
015 \* limitations under the License.  
016 \*/  
017  
018package org.apache.commons.beanutils.locale.converters;  
019  
020import org.apache.commons.beanutils.locale.BaseLocaleConverter;  
021import org.apache.commons.logging.Log;  
022import org.apache.commons.logging.LogFactory;  
023  
024import java.math.BigDecimal;  
025import java.math.BigInteger;  
026import java.text.DecimalFormat;  
027import java.text.NumberFormat;  
028import java.text.ParseException;  
029import java.text.SimpleDateFormat;  
030import java.util.Date;  
031import java.util.Locale;  
032  
033  
034/\*\*  
035 \* <p>Standard {@link org.apache.commons.beanutils.locale.LocaleConverter}  
036 \* implementation that converts an incoming  
037 \* locale-sensitive object into a <code>java.lang.String</code> object,  
038 \* optionally using a default value or throwing a  
039 \* {@link org.apache.commons.beanutils.ConversionException}  
040 \* if a conversion error occurs.</p>  
041 \*  
042 \* @version $Id$  
043 \*/  
044  
045public class StringLocaleConverter extends BaseLocaleConverter {  
046  
047 // ----------------------------------------------------- Instance Variables  
048  
049 /\*\* All logging goes through this logger \*/  
050 private final Log log = LogFactory.getLog(StringLocaleConverter.class); //msz fix  
051  
052  
053 // ----------------------------------------------------------- Constructors  
054  
055 /\*\*  
056 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
057 \* that will throw a {@link org.apache.commons.beanutils.ConversionException}  
058 \* if a conversion error occurs. The locale is the default locale for  
059 \* this instance of the Java Virtual Machine and an unlocalized pattern is used  
060 \* for the convertion.  
061 \*  
062 \*/  
063 public StringLocaleConverter() {  
064  
065 this(false);  
066 }  
067  
068 /\*\*  
069 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
070 \* that will throw a {@link org.apache.commons.beanutils.ConversionException}  
071 \* if a conversion error occurs. The locale is the default locale for  
072 \* this instance of the Java Virtual Machine.  
073 \*  
074 \* @param locPattern Indicate whether the pattern is localized or not  
075 \*/  
076 public StringLocaleConverter(final boolean locPattern) {  
077  
078 this(Locale.getDefault(), locPattern);  
079 }  
080  
081 /\*\*  
082 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
083 \* that will throw a {@link org.apache.commons.beanutils.ConversionException}  
084 \* if a conversion error occurs. An unlocalized pattern is used for the convertion.  
085 \*  
086 \* @param locale The locale  
087 \*/  
088 public StringLocaleConverter(final Locale locale) {  
089  
090 this(locale, false);  
091 }  
092  
093 /\*\*  
094 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
095 \* that will throw a {@link org.apache.commons.beanutils.ConversionException}  
096 \* if a conversion error occurs.  
097 \*  
098 \* @param locale The locale  
099 \* @param locPattern Indicate whether the pattern is localized or not  
100 \*/  
101 public StringLocaleConverter(final Locale locale, final boolean locPattern) {  
102  
103 this(locale, (String) null, locPattern);  
104 }  
105  
106 /\*\*  
107 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
108 \* that will throw a {@link org.apache.commons.beanutils.ConversionException}  
109 \* if a conversion error occurs. An unlocalized pattern is used for the convertion.  
110 \*  
111 \* @param locale The locale  
112 \* @param pattern The convertion pattern  
113 \*/  
114 public StringLocaleConverter(final Locale locale, final String pattern) {  
115  
116 this(locale, pattern, false);  
117 }  
118  
119 /\*\*  
120 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
121 \* that will throw a {@link org.apache.commons.beanutils.ConversionException}  
122 \* if a conversion error occurs.  
123 \*  
124 \* @param locale The locale  
125 \* @param pattern The convertion pattern  
126 \* @param locPattern Indicate whether the pattern is localized or not  
127 \*/  
128 public StringLocaleConverter(final Locale locale, final String pattern, final boolean locPattern) {  
129  
130 super(locale, pattern, locPattern);  
131 }  
132  
133 /\*\*  
134 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
135 \* that will return the specified default value  
136 \* if a conversion error occurs. The locale is the default locale for  
137 \* this instance of the Java Virtual Machine and an unlocalized pattern is used  
138 \* for the convertion.  
139 \*  
140 \* @param defaultValue The default value to be returned  
141 \*/  
142 public StringLocaleConverter(final Object defaultValue) {  
143  
144 this(defaultValue, false);  
145 }  
146  
147 /\*\*  
148 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
149 \* that will return the specified default value  
150 \* if a conversion error occurs. The locale is the default locale for  
151 \* this instance of the Java Virtual Machine.  
152 \*  
153 \* @param defaultValue The default value to be returned  
154 \* @param locPattern Indicate whether the pattern is localized or not  
155 \*/  
156 public StringLocaleConverter(final Object defaultValue, final boolean locPattern) {  
157  
158 this(defaultValue, Locale.getDefault(), locPattern);  
159 }  
160  
161 /\*\*  
162 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
163 \* that will return the specified default value  
164 \* if a conversion error occurs. An unlocalized pattern is used for the convertion.  
165 \*  
166 \* @param defaultValue The default value to be returned  
167 \* @param locale The locale  
168 \*/  
169 public StringLocaleConverter(final Object defaultValue, final Locale locale) {  
170  
171 this(defaultValue, locale, false);  
172 }  
173  
174 /\*\*  
175 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
176 \* that will return the specified default value  
177 \* if a conversion error occurs.  
178 \*  
179 \* @param defaultValue The default value to be returned  
180 \* @param locale The locale  
181 \* @param locPattern Indicate whether the pattern is localized or not  
182 \*/  
183 public StringLocaleConverter(final Object defaultValue, final Locale locale, final boolean locPattern) {  
184  
185 this(defaultValue, locale, null, locPattern);  
186 }  
187  
188 /\*\*  
189 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
190 \* that will return the specified default value  
191 \* if a conversion error occurs. An unlocalized pattern is used for the convertion.  
192 \*  
193 \* @param defaultValue The default value to be returned  
194 \* @param locale The locale  
195 \* @param pattern The convertion pattern  
196 \*/  
197 public StringLocaleConverter(final Object defaultValue, final Locale locale, final String pattern) {  
198  
199 this(defaultValue, locale, pattern, false);  
200 }  
201  
202 /\*\*  
203 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
204 \* that will return the specified default value  
205 \* if a conversion error occurs.  
206 \*  
207 \* @param defaultValue The default value to be returned  
208 \* @param locale The locale  
209 \* @param pattern The convertion pattern  
210 \* @param locPattern Indicate whether the pattern is localized or not  
211 \*/  
212 public StringLocaleConverter(final Object defaultValue, final Locale locale, final String pattern, final boolean locPattern) {  
213  
214 super(defaultValue, locale, pattern, locPattern);  
215 }  
216  
217 // --------------------------------------------------------- Methods  
218  
219 /\*\*  
220 \* Convert the specified locale-sensitive input object into an output object of the  
221 \* specified type.  
222 \*  
223 \* @param value The input object to be converted  
224 \* @param pattern The pattern is used for the convertion  
225 \* @return The converted value  
226 \*  
227 \* @throws org.apache.commons.beanutils.ConversionException if conversion  
228 \* cannot be performed successfully  
229 \* @throws ParseException if an error occurs  
230 \*/  
231 @Override  
232 protected Object parse(final Object value, final String pattern) throws ParseException {  
233  
234 String result = null;  
235  
236 if ((value instanceof Integer) ||  
237 (value instanceof Long) ||  
238 (value instanceof BigInteger) ||  
239 (value instanceof Byte) ||  
240 (value instanceof Short)) {  
241  
242 result = getDecimalFormat(locale, pattern).format(((Number) value).longValue());  
243 }  
244 else if ((value instanceof Double) ||  
245 (value instanceof BigDecimal) ||  
246 (value instanceof Float)) {  
247  
248 result = getDecimalFormat(locale, pattern).format(((Number) value).doubleValue());  
249 }  
250 else if (value instanceof Date) { // java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp  
251  
252 final SimpleDateFormat dateFormat =  
253 new SimpleDateFormat(pattern, locale);  
254  
255 result = dateFormat.format(value);  
256 }  
257 else {  
258 result = value.toString();  
259 }  
260  
261 return result;  
262 }  
263  
264 /\*\*  
265 \* Make an instance of DecimalFormat.  
266 \*  
267 \* @param locale The locale  
268 \* @param pattern The pattern is used for the convertion  
269 \* @return The format for the locale and pattern  
270 \*  
271 \* @throws ConversionException if conversion cannot be performed  
272 \* successfully  
273 \* @throws ParseException if an error occurs parsing a String to a Number  
274 \*/  
275 private DecimalFormat getDecimalFormat(final Locale locale, final String pattern) {  
276  
277 final DecimalFormat numberFormat = (DecimalFormat) NumberFormat.getInstance(locale);  
278  
279 // if some constructors default pattern to null, it makes only sense to handle null pattern gracefully  
280 if (pattern != null) {  
281 if (locPattern) {  
282 numberFormat.applyLocalizedPattern(pattern);  
283 } else {  
284 numberFormat.applyPattern(pattern);  
285 }  
286 } else {  
287 log.debug("No pattern provided, using default.");  
288 }  
289  
290 return numberFormat;  
291 }  
292}