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 java.text.DateFormat;  
021import java.text.DateFormatSymbols;  
022import java.text.ParseException;  
023import java.text.ParsePosition;  
024import java.text.SimpleDateFormat;  
025import java.util.Locale;  
026  
027import org.apache.commons.beanutils.ConversionException;  
028import org.apache.commons.beanutils.locale.BaseLocaleConverter;  
029import org.apache.commons.logging.Log;  
030import org.apache.commons.logging.LogFactory;  
031  
032  
033/\*\*  
034 \* <p>Standard {@link org.apache.commons.beanutils.locale.LocaleConverter}  
035 \* implementation that converts an incoming  
036 \* locale-sensitive String into a <code>java.util.Date</code> object,  
037 \* optionally using a default value or throwing a  
038 \* {@link org.apache.commons.beanutils.ConversionException}  
039 \* if a conversion error occurs.</p>  
040 \*  
041 \* @version $Id$  
042 \*/  
043  
044public class DateLocaleConverter extends BaseLocaleConverter {  
045  
046 // ----------------------------------------------------- Instance Variables  
047  
048 /\*\* All logging goes through this logger \*/  
049 private final Log log = LogFactory.getLog(DateLocaleConverter.class);  
050  
051 /\*\* Should the date conversion be lenient? \*/  
052 boolean isLenient = false;  
053  
054 /\*\*  
055 \* Default Pattern Characters  
056 \*  
057 \*/  
058 private static final String DEFAULT\_PATTERN\_CHARS = DateLocaleConverter.initDefaultChars();  
059  
060 // ----------------------------------------------------------- Constructors  
061  
062 /\*\*  
063 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
064 \* that will throw a {@link org.apache.commons.beanutils.ConversionException}  
065 \* if a conversion error occurs. The locale is the default locale for  
066 \* this instance of the Java Virtual Machine and an unlocalized pattern is used  
067 \* for the convertion.  
068 \*  
069 \*/  
070 public DateLocaleConverter() {  
071  
072 this(false);  
073 }  
074  
075 /\*\*  
076 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
077 \* that will throw a {@link org.apache.commons.beanutils.ConversionException}  
078 \* if a conversion error occurs. The locale is the default locale for  
079 \* this instance of the Java Virtual Machine.  
080 \*  
081 \* @param locPattern Indicate whether the pattern is localized or not  
082 \*/  
083 public DateLocaleConverter(final boolean locPattern) {  
084  
085 this(Locale.getDefault(), locPattern);  
086 }  
087  
088 /\*\*  
089 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
090 \* that will throw a {@link org.apache.commons.beanutils.ConversionException}  
091 \* if a conversion error occurs. An unlocalized pattern is used for the convertion.  
092 \*  
093 \* @param locale The locale  
094 \*/  
095 public DateLocaleConverter(final Locale locale) {  
096  
097 this(locale, false);  
098 }  
099  
100 /\*\*  
101 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
102 \* that will throw a {@link org.apache.commons.beanutils.ConversionException}  
103 \* if a conversion error occurs.  
104 \*  
105 \* @param locale The locale  
106 \* @param locPattern Indicate whether the pattern is localized or not  
107 \*/  
108 public DateLocaleConverter(final Locale locale, final boolean locPattern) {  
109  
110 this(locale, (String) null, locPattern);  
111 }  
112  
113 /\*\*  
114 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
115 \* that will throw a {@link org.apache.commons.beanutils.ConversionException}  
116 \* if a conversion error occurs. An unlocalized pattern is used for the convertion.  
117 \*  
118 \* @param locale The locale  
119 \* @param pattern The convertion pattern  
120 \*/  
121 public DateLocaleConverter(final Locale locale, final String pattern) {  
122  
123 this(locale, pattern, false);  
124 }  
125  
126 /\*\*  
127 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
128 \* that will throw a {@link org.apache.commons.beanutils.ConversionException}  
129 \* if a conversion error occurs.  
130 \*  
131 \* @param locale The locale  
132 \* @param pattern The convertion pattern  
133 \* @param locPattern Indicate whether the pattern is localized or not  
134 \*/  
135 public DateLocaleConverter(final Locale locale, final String pattern, final boolean locPattern) {  
136  
137 super(locale, pattern, locPattern);  
138 }  
139  
140 /\*\*  
141 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
142 \* that will return the specified default value  
143 \* if a conversion error occurs. The locale is the default locale for  
144 \* this instance of the Java Virtual Machine and an unlocalized pattern is used  
145 \* for the convertion.  
146 \*  
147 \* @param defaultValue The default value to be returned  
148 \*/  
149 public DateLocaleConverter(final Object defaultValue) {  
150  
151 this(defaultValue, false);  
152 }  
153  
154 /\*\*  
155 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
156 \* that will return the specified default value  
157 \* if a conversion error occurs. The locale is the default locale for  
158 \* this instance of the Java Virtual Machine.  
159 \*  
160 \* @param defaultValue The default value to be returned  
161 \* @param locPattern Indicate whether the pattern is localized or not  
162 \*/  
163 public DateLocaleConverter(final Object defaultValue, final boolean locPattern) {  
164  
165 this(defaultValue, Locale.getDefault(), locPattern);  
166 }  
167  
168 /\*\*  
169 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
170 \* that will return the specified default value  
171 \* if a conversion error occurs. An unlocalized pattern is used for the convertion.  
172 \*  
173 \* @param defaultValue The default value to be returned  
174 \* @param locale The locale  
175 \*/  
176 public DateLocaleConverter(final Object defaultValue, final Locale locale) {  
177  
178 this(defaultValue, locale, false);  
179 }  
180  
181 /\*\*  
182 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
183 \* that will return the specified default value  
184 \* if a conversion error occurs.  
185 \*  
186 \* @param defaultValue The default value to be returned  
187 \* @param locale The locale  
188 \* @param locPattern Indicate whether the pattern is localized or not  
189 \*/  
190 public DateLocaleConverter(final Object defaultValue, final Locale locale, final boolean locPattern) {  
191  
192 this(defaultValue, locale, null, locPattern);  
193 }  
194  
195  
196 /\*\*  
197 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
198 \* that will return the specified default value  
199 \* if a conversion error occurs. An unlocalized pattern is used for the convertion.  
200 \*  
201 \* @param defaultValue The default value to be returned  
202 \* @param locale The locale  
203 \* @param pattern The convertion pattern  
204 \*/  
205 public DateLocaleConverter(final Object defaultValue, final Locale locale, final String pattern) {  
206  
207 this(defaultValue, locale, pattern, false);  
208 }  
209  
210 /\*\*  
211 \* Create a {@link org.apache.commons.beanutils.locale.LocaleConverter}  
212 \* that will return the specified default value  
213 \* if a conversion error occurs.  
214 \*  
215 \* @param defaultValue The default value to be returned  
216 \* @param locale The locale  
217 \* @param pattern The convertion pattern  
218 \* @param locPattern Indicate whether the pattern is localized or not  
219 \*/  
220 public DateLocaleConverter(final Object defaultValue, final Locale locale, final String pattern, final boolean locPattern) {  
221  
222 super(defaultValue, locale, pattern, locPattern);  
223 }  
224  
225 // --------------------------------------------------------- Methods  
226  
227 /\*\*  
228 \* Returns whether date formatting is lenient.  
229 \*  
230 \* @return true if the <code>DateFormat</code> used for formatting is lenient  
231 \* @see java.text.DateFormat#isLenient  
232 \*/  
233 public boolean isLenient() {  
234 return isLenient;  
235 }  
236  
237 /\*\*  
238 \* Specify whether or not date-time parsing should be lenient.  
239 \*  
240 \* @param lenient true if the <code>DateFormat</code> used for formatting should be lenient  
241 \* @see java.text.DateFormat#setLenient  
242 \*/  
243 public void setLenient(final boolean lenient) {  
244 isLenient = lenient;  
245 }  
246  
247 // --------------------------------------------------------- Methods  
248  
249 /\*\*  
250 \* Convert the specified locale-sensitive input object into an output object of the  
251 \* specified type.  
252 \*  
253 \* @param value The input object to be converted  
254 \* @param pattern The pattern is used for the convertion  
255 \* @return the converted Date value  
256 \*  
257 \* @throws org.apache.commons.beanutils.ConversionException  
258 \* if conversion cannot be performed successfully  
259 \* @throws ParseException if an error occurs parsing  
260 \*/  
261 @Override  
262 protected Object parse(final Object value, String pattern) throws ParseException {  
263  
264 // Handle Date  
265 if (value instanceof java.util.Date) {  
266 return value;  
267 }  
268  
269 // Handle Calendar  
270 if (value instanceof java.util.Calendar) {  
271 return ((java.util.Calendar)value).getTime();  
272 }  
273  
274 if (locPattern) {  
275 pattern = convertLocalizedPattern(pattern, locale);  
276 }  
277  
278 // Create Formatter - use default if pattern is null  
279 final DateFormat formatter = pattern == null ? DateFormat.getDateInstance(DateFormat.SHORT, locale)  
280 : new SimpleDateFormat(pattern, locale);  
281 formatter.setLenient(isLenient);  
282  
283  
284 // Parse the Date  
285 final ParsePosition pos = new ParsePosition(0);  
286 final String strValue = value.toString();  
287 final Object parsedValue = formatter.parseObject(strValue, pos);  
288 if (pos.getErrorIndex() > -1) {  
289 throw new ConversionException("Error parsing date '" + value +  
290 "' at position="+ pos.getErrorIndex());  
291 }  
292 if (pos.getIndex() < strValue.length()) {  
293 throw new ConversionException("Date '" + value +  
294 "' contains unparsed characters from position=" + pos.getIndex());  
295 }  
296  
297 return parsedValue;  
298 }  
299  
300 /\*\*  
301 \* Convert a pattern from a localized format to the default format.  
302 \*  
303 \* @param locale The locale  
304 \* @param localizedPattern The pattern in 'local' symbol format  
305 \* @return pattern in 'default' symbol format  
306 \*/  
307 private String convertLocalizedPattern(final String localizedPattern, final Locale locale) {  
308  
309 if (localizedPattern == null) {  
310 return null;  
311 }  
312  
313 // Note that this is a little obtuse.  
314 // However, it is the best way that anyone can come up with  
315 // that works with some 1.4 series JVM.  
316  
317 // Get the symbols for the localized pattern  
318 final DateFormatSymbols localizedSymbols = new DateFormatSymbols(locale);  
319 final String localChars = localizedSymbols.getLocalPatternChars();  
320  
321 if (DEFAULT\_PATTERN\_CHARS.equals(localChars)) {  
322 return localizedPattern;  
323 }  
324  
325 // Convert the localized pattern to default  
326 String convertedPattern = null;  
327 try {  
328 convertedPattern = convertPattern(localizedPattern,  
329 localChars,  
330 DEFAULT\_PATTERN\_CHARS);  
331 } catch (final Exception ex) {  
332 log.debug("Converting pattern '" + localizedPattern + "' for " + locale, ex);  
333 }  
334 return convertedPattern;  
335 }  
336  
337 /\*\*  
338 \* <p>Converts a Pattern from one character set to another.</p>  
339 \*/  
340 private String convertPattern(final String pattern, final String fromChars, final String toChars) {  
341  
342 final StringBuilder converted = new StringBuilder();  
343 boolean quoted = false;  
344  
345 for (int i = 0; i < pattern.length(); ++i) {  
346 char thisChar = pattern.charAt(i);  
347 if (quoted) {  
348 if (thisChar == '\'') {  
349 quoted = false;  
350 }  
351 } else {  
352 if (thisChar == '\'') {  
353 quoted = true;  
354 } else if ((thisChar >= 'a' && thisChar <= 'z') ||  
355 (thisChar >= 'A' && thisChar <= 'Z')) {  
356 final int index = fromChars.indexOf(thisChar );  
357 if (index == -1) {  
358 throw new IllegalArgumentException(  
359 "Illegal pattern character '" + thisChar + "'");  
360 }  
361 thisChar = toChars.charAt(index);  
362 }  
363 }  
364 converted.append(thisChar);  
365 }  
366  
367 if (quoted) {  
368 throw new IllegalArgumentException("Unfinished quote in pattern");  
369 }  
370  
371 return converted.toString();  
372 }  
373  
374 /\*\*  
375 \* This method is called at class initialization time to define the  
376 \* value for constant member DEFAULT\_PATTERN\_CHARS. All other methods needing  
377 \* this data should just read that constant.  
378 \*/  
379 private static String initDefaultChars() {  
380 final DateFormatSymbols defaultSymbols = new DateFormatSymbols(Locale.US);  
381 return defaultSymbols.getLocalPatternChars();  
382 }  
383  
384}