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;  
019  
020  
021import java.lang.reflect.InvocationTargetException;  
022import java.util.Locale;  
023  
024import org.apache.commons.beanutils.BeanUtils;  
025  
026  
027  
028/\*\*  
029 \* <p>Utility methods for populating JavaBeans properties  
030 \* via reflection in a locale-dependent manner.</p>  
031 \*  
032 \* <p>The implementations for these methods are provided by <code>LocaleBeanUtilsBean</code>.  
033 \* For more details see {@link LocaleBeanUtilsBean}.</p>  
034 \*  
035 \* @version $Id$  
036 \*/  
037  
038public class LocaleBeanUtils extends BeanUtils {  
039  
040  
041 // ----------------------------------------------------- Instance Variables  
042  
043 /\*\*  
044 \* <p>Gets the locale used when no locale is passed.</p>  
045 \*  
046 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
047 \*  
048 \* @return the default locale  
049 \* @see LocaleBeanUtilsBean#getDefaultLocale()  
050 \*/  
051 public static Locale getDefaultLocale() {  
052  
053 return LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().getDefaultLocale();  
054 }  
055  
056  
057 /\*\*  
058 \* <p>Sets the locale used when no locale is passed.</p>  
059 \*  
060 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
061 \*  
062 \* @param locale the default locale  
063 \* @see LocaleBeanUtilsBean#setDefaultLocale(Locale)  
064 \*/  
065 public static void setDefaultLocale(final Locale locale) {  
066  
067 LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().setDefaultLocale(locale);  
068 }  
069  
070 /\*\*  
071 \* <p>Gets whether the pattern is localized or not.</p>  
072 \*  
073 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
074 \*  
075 \* @return <code>true</code> if pattern is localized,  
076 \* otherwise <code>false</code>  
077 \* @see LocaleBeanUtilsBean#getApplyLocalized()  
078 \*/  
079 public static boolean getApplyLocalized() {  
080  
081 return LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().getApplyLocalized();  
082 }  
083  
084 /\*\*  
085 \* <p>Sets whether the pattern is localized or not.</p>  
086 \*  
087 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
088 \*  
089 \* @param newApplyLocalized <code>true</code> if pattern is localized,  
090 \* otherwise <code>false</code>  
091 \* @see LocaleBeanUtilsBean#setApplyLocalized(boolean)  
092 \*/  
093 public static void setApplyLocalized(final boolean newApplyLocalized) {  
094  
095 LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().setApplyLocalized(newApplyLocalized);  
096 }  
097  
098  
099 // --------------------------------------------------------- Public Methods  
100  
101 /\*\*  
102 \* <p>Return the value of the specified locale-sensitive indexed property  
103 \* of the specified bean, as a String.</p>  
104 \*  
105 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
106 \*  
107 \* @param bean Bean whose property is to be extracted  
108 \* @param name <code>propertyname[index]</code> of the property value  
109 \* to be extracted  
110 \* @param pattern The conversion pattern  
111 \* @return The indexed property's value, converted to a String  
112 \*  
113 \* @throws IllegalAccessException if the caller does not have  
114 \* access to the property accessor method  
115 \* @throws InvocationTargetException if the property accessor method  
116 \* throws an exception  
117 \* @throws NoSuchMethodException if an accessor method for this  
118 \* propety cannot be found  
119 \*  
120 \* @see LocaleBeanUtilsBean#getIndexedProperty(Object, String, String)  
121 \*/  
122 public static String getIndexedProperty(final Object bean, final String name, final String pattern)  
123 throws IllegalAccessException, InvocationTargetException,  
124 NoSuchMethodException {  
125  
126 return LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().getIndexedProperty(bean, name, pattern);  
127 }  
128  
129 /\*\*  
130 \* Return the value of the specified locale-sensitive indexed property  
131 \* of the specified bean, as a String using the default conversion pattern of  
132 \* the corresponding {@link LocaleConverter}.  
133 \*  
134 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
135 \*  
136 \* @param bean Bean whose property is to be extracted  
137 \* @param name <code>propertyname[index]</code> of the property value  
138 \* to be extracted  
139 \* @return The indexed property's value, converted to a String  
140 \*  
141 \* @throws IllegalAccessException if the caller does not have  
142 \* access to the property accessor method  
143 \* @throws InvocationTargetException if the property accessor method  
144 \* throws an exception  
145 \* @throws NoSuchMethodException if an accessor method for this  
146 \* propety cannot be found  
147 \*  
148 \* @see LocaleBeanUtilsBean#getIndexedProperty(Object, String)  
149 \*/  
150 public static String getIndexedProperty(final Object bean, final String name)  
151 throws IllegalAccessException, InvocationTargetException,  
152 NoSuchMethodException {  
153  
154 return LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().getIndexedProperty(bean, name);  
155 }  
156  
157 /\*\*  
158 \* <p>Return the value of the specified locale-sensetive indexed property  
159 \* of the specified bean, as a String using the specified conversion pattern.</p>  
160 \*  
161 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
162 \*  
163 \* @param bean Bean whose property is to be extracted  
164 \* @param name Simple property name of the property value to be extracted  
165 \* @param index Index of the property value to be extracted  
166 \* @param pattern The conversion pattern  
167 \* @return The indexed property's value, converted to a String  
168 \*  
169 \* @throws IllegalAccessException if the caller does not have  
170 \* access to the property accessor method  
171 \* @throws InvocationTargetException if the property accessor method  
172 \* throws an exception  
173 \* @throws NoSuchMethodException if an accessor method for this  
174 \* propety cannot be found  
175 \*  
176 \* @see LocaleBeanUtilsBean#getIndexedProperty(Object, String, int, String)  
177 \*/  
178 public static String getIndexedProperty(final Object bean,  
179 final String name, final int index, final String pattern)  
180 throws IllegalAccessException, InvocationTargetException,  
181 NoSuchMethodException {  
182  
183 return LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().getIndexedProperty(bean, name, index, pattern);  
184 }  
185  
186 /\*\*  
187 \* <p>Return the value of the specified locale-sensetive indexed property  
188 \* of the specified bean, as a String using the default conversion pattern of  
189 \* the corresponding {@link LocaleConverter}.</p>  
190 \*  
191 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
192 \*  
193 \* @param bean Bean whose property is to be extracted  
194 \* @param name Simple property name of the property value to be extracted  
195 \* @param index Index of the property value to be extracted  
196 \* @return The indexed property's value, converted to a String  
197 \*  
198 \* @throws IllegalAccessException if the caller does not have  
199 \* access to the property accessor method  
200 \* @throws InvocationTargetException if the property accessor method  
201 \* throws an exception  
202 \* @throws NoSuchMethodException if an accessor method for this  
203 \* propety cannot be found  
204 \*  
205 \* @see LocaleBeanUtilsBean#getIndexedProperty(Object, String, int)  
206 \*/  
207 public static String getIndexedProperty(final Object bean,  
208 final String name, final int index)  
209 throws IllegalAccessException, InvocationTargetException,  
210 NoSuchMethodException {  
211 return LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().getIndexedProperty(bean, name, index);  
212 }  
213  
214 /\*\*  
215 \* <p>Return the value of the specified simple locale-sensitive property  
216 \* of the specified bean, converted to a String using the specified  
217 \* conversion pattern.</p>  
218 \*  
219 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
220 \*  
221 \* @param bean Bean whose property is to be extracted  
222 \* @param name Name of the property to be extracted  
223 \* @param pattern The conversion pattern  
224 \* @return The property's value, converted to a String  
225 \*  
226 \* @throws IllegalAccessException if the caller does not have  
227 \* access to the property accessor method  
228 \* @throws InvocationTargetException if the property accessor method  
229 \* throws an exception  
230 \* @throws NoSuchMethodException if an accessor method for this  
231 \* propety cannot be found  
232 \*  
233 \* @see LocaleBeanUtilsBean#getSimpleProperty(Object, String, String)  
234 \*/  
235 public static String getSimpleProperty(final Object bean, final String name, final String pattern)  
236 throws IllegalAccessException, InvocationTargetException,  
237 NoSuchMethodException {  
238  
239 return LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().getSimpleProperty(bean, name, pattern);  
240 }  
241  
242 /\*\*  
243 \* <p>Return the value of the specified simple locale-sensitive property  
244 \* of the specified bean, converted to a String using the default  
245 \* conversion pattern of the corresponding {@link LocaleConverter}.</p>  
246 \*  
247 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
248 \*  
249 \* @param bean Bean whose property is to be extracted  
250 \* @param name Name of the property to be extracted  
251 \* @return The property's value, converted to a String  
252 \*  
253 \* @throws IllegalAccessException if the caller does not have  
254 \* access to the property accessor method  
255 \* @throws InvocationTargetException if the property accessor method  
256 \* throws an exception  
257 \* @throws NoSuchMethodException if an accessor method for this  
258 \* propety cannot be found  
259 \*  
260 \* @see LocaleBeanUtilsBean#getSimpleProperty(Object, String)  
261 \*/  
262 public static String getSimpleProperty(final Object bean, final String name)  
263 throws IllegalAccessException, InvocationTargetException,  
264 NoSuchMethodException {  
265  
266 return LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().getSimpleProperty(bean, name);  
267 }  
268  
269 /\*\*  
270 \* <p>Return the value of the specified mapped locale-sensitive property  
271 \* of the specified bean, as a String using the specified conversion pattern.</p>  
272 \*  
273 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
274 \*  
275 \* @param bean Bean whose property is to be extracted  
276 \* @param name Simple property name of the property value to be extracted  
277 \* @param key Lookup key of the property value to be extracted  
278 \* @param pattern The conversion pattern  
279 \* @return The mapped property's value, converted to a String  
280 \*  
281 \* @throws IllegalAccessException if the caller does not have  
282 \* access to the property accessor method  
283 \* @throws InvocationTargetException if the property accessor method  
284 \* throws an exception  
285 \* @throws NoSuchMethodException if an accessor method for this  
286 \* propety cannot be found  
287 \*  
288 \* @see LocaleBeanUtilsBean#getMappedProperty(Object, String, String, String)  
289 \*/  
290 public static String getMappedProperty(final Object bean,  
291 final String name, final String key, final String pattern)  
292 throws IllegalAccessException, InvocationTargetException,  
293 NoSuchMethodException {  
294  
295 return LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().getMappedProperty(bean, name, key, pattern);  
296 }  
297  
298 /\*\*  
299 \* <p>Return the value of the specified mapped locale-sensitive property  
300 \* of the specified bean, as a String  
301 \* The key is specified as a method parameter and must \*not\* be included  
302 \* in the property name expression.</p>  
303 \*  
304 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
305 \*  
306 \* @param bean Bean whose property is to be extracted  
307 \* @param name Simple property name of the property value to be extracted  
308 \* @param key Lookup key of the property value to be extracted  
309 \* @return The mapped property's value, converted to a String  
310 \*  
311 \* @throws IllegalAccessException if the caller does not have  
312 \* access to the property accessor method  
313 \* @throws InvocationTargetException if the property accessor method  
314 \* throws an exception  
315 \* @throws NoSuchMethodException if an accessor method for this  
316 \* propety cannot be found  
317 \*  
318 \* @see LocaleBeanUtilsBean#getMappedProperty(Object, String, String)  
319 \*/  
320 public static String getMappedProperty(final Object bean,  
321 final String name, final String key)  
322 throws IllegalAccessException, InvocationTargetException,  
323 NoSuchMethodException {  
324  
325 return LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().getMappedProperty(bean, name, key);  
326 }  
327  
328  
329 /\*\*  
330 \* <p>Return the value of the specified locale-sensitive mapped property  
331 \* of the specified bean, as a String using the specified pattern.</p>  
332 \*  
333 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
334 \*  
335 \* @param bean Bean whose property is to be extracted  
336 \* @param name <code>propertyname(index)</code> of the property value  
337 \* to be extracted  
338 \* @param pattern The conversion pattern  
339 \* @return The mapped property's value, converted to a String  
340 \*  
341 \* @throws IllegalAccessException if the caller does not have  
342 \* access to the property accessor method  
343 \* @throws InvocationTargetException if the property accessor method  
344 \* throws an exception  
345 \* @throws NoSuchMethodException if an accessor method for this  
346 \* propety cannot be found  
347 \*  
348 \* @see LocaleBeanUtilsBean#getMappedPropertyLocale(Object, String, String)  
349 \*/  
350 public static String getMappedPropertyLocale(final Object bean, final String name, final String pattern)  
351 throws IllegalAccessException, InvocationTargetException,  
352 NoSuchMethodException {  
353  
354 return LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().getMappedPropertyLocale(bean, name, pattern);  
355 }  
356  
357  
358 /\*\*  
359 \* <p>Return the value of the specified locale-sensitive mapped property  
360 \* of the specified bean, as a String using the default  
361 \* conversion pattern of the corresponding {@link LocaleConverter}.</p>  
362 \*  
363 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
364 \*  
365 \* @param bean Bean whose property is to be extracted  
366 \* @param name <code>propertyname(index)</code> of the property value  
367 \* to be extracted  
368 \* @return The mapped property's value, converted to a String  
369 \*  
370 \* @throws IllegalAccessException if the caller does not have  
371 \* access to the property accessor method  
372 \* @throws InvocationTargetException if the property accessor method  
373 \* throws an exception  
374 \* @throws NoSuchMethodException if an accessor method for this  
375 \* propety cannot be found  
376 \*  
377 \* @see LocaleBeanUtilsBean#getMappedProperty(Object, String)  
378 \*/  
379 public static String getMappedProperty(final Object bean, final String name)  
380 throws IllegalAccessException, InvocationTargetException,  
381 NoSuchMethodException {  
382  
383 return LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().getMappedProperty(bean, name);  
384 }  
385  
386 /\*\*  
387 \* <p>Return the value of the (possibly nested) locale-sensitive property  
388 \* of the specified name, for the specified bean,  
389 \* as a String using the specified pattern.</p>  
390 \*  
391 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
392 \*  
393 \* @param bean Bean whose property is to be extracted  
394 \* @param name Possibly nested name of the property to be extracted  
395 \* @param pattern The conversion pattern  
396 \* @return The nested property's value, converted to a String  
397 \*  
398 \* @throws IllegalAccessException if the caller does not have  
399 \* access to the property accessor method  
400 \* @throws InvocationTargetException if the property accessor method  
401 \* throws an exception  
402 \* @throws NoSuchMethodException if an accessor method for this  
403 \* propety cannot be found  
404 \*  
405 \* @see LocaleBeanUtilsBean#getNestedProperty(Object, String, String)  
406 \*/  
407 public static String getNestedProperty(final Object bean, final String name, final String pattern)  
408 throws IllegalAccessException, InvocationTargetException,  
409 NoSuchMethodException {  
410  
411 return LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().getNestedProperty(bean, name, pattern);  
412 }  
413  
414 /\*\*  
415 \* <p>Return the value of the (possibly nested) locale-sensitive property  
416 \* of the specified name.</p>  
417 \*  
418 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
419 \*  
420 \* @param bean Bean whose property is to be extracted  
421 \* @param name Possibly nested name of the property to be extracted  
422 \* @return The nested property's value, converted to a String  
423 \*  
424 \* @throws IllegalAccessException if the caller does not have  
425 \* access to the property accessor method  
426 \* @throws InvocationTargetException if the property accessor method  
427 \* throws an exception  
428 \* @throws NoSuchMethodException if an accessor method for this  
429 \* propety cannot be found  
430 \*  
431 \* @see LocaleBeanUtilsBean#getNestedProperty(Object, String)  
432 \*/  
433 public static String getNestedProperty(final Object bean, final String name)  
434 throws IllegalAccessException, InvocationTargetException,  
435 NoSuchMethodException {  
436  
437 return LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().getNestedProperty(bean, name);  
438 }  
439  
440 /\*\*  
441 \* <p>Return the value of the specified locale-sensitive property  
442 \* of the specified bean.</p>  
443 \*  
444 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
445 \*  
446 \* @param bean Bean whose property is to be extracted  
447 \* @param name Possibly indexed and/or nested name of the property  
448 \* to be extracted  
449 \* @param pattern The conversion pattern  
450 \* @return The nested property's value, converted to a String  
451 \*  
452 \* @throws IllegalAccessException if the caller does not have  
453 \* access to the property accessor method  
454 \* @throws InvocationTargetException if the property accessor method  
455 \* throws an exception  
456 \* @throws NoSuchMethodException if an accessor method for this  
457 \* propety cannot be found  
458 \*  
459 \* @see LocaleBeanUtilsBean#getProperty(Object, String, String)  
460 \*/  
461 public static String getProperty(final Object bean, final String name, final String pattern)  
462 throws IllegalAccessException, InvocationTargetException,  
463 NoSuchMethodException {  
464  
465 return LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().getProperty(bean, name, pattern);  
466 }  
467  
468 /\*\*  
469 \* <p>Return the value of the specified locale-sensitive property  
470 \* of the specified bean.</p>  
471 \*  
472 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
473 \*  
474 \* @param bean Bean whose property is to be extracted  
475 \* @param name Possibly indexed and/or nested name of the property  
476 \* to be extracted  
477 \* @return The property's value, converted to a String  
478 \*  
479 \* @throws IllegalAccessException if the caller does not have  
480 \* access to the property accessor method  
481 \* @throws InvocationTargetException if the property accessor method  
482 \* throws an exception  
483 \* @throws NoSuchMethodException if an accessor method for this  
484 \* propety cannot be found  
485 \*  
486 \* @see LocaleBeanUtilsBean#getProperty(Object, String)  
487 \*/  
488 public static String getProperty(final Object bean, final String name)  
489 throws IllegalAccessException, InvocationTargetException,  
490 NoSuchMethodException {  
491  
492 return LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().getProperty(bean, name);  
493 }  
494  
495 /\*\*  
496 \* <p>Set the specified locale-sensitive property value, performing type  
497 \* conversions as required to conform to the type of the destination property  
498 \* using the default conversion pattern of the corresponding {@link LocaleConverter}.</p>  
499 \*  
500 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
501 \*  
502 \* @param bean Bean on which setting is to be performed  
503 \* @param name Property name (can be nested/indexed/mapped/combo)  
504 \* @param value Value to be set  
505 \*  
506 \* @throws IllegalAccessException if the caller does not have  
507 \* access to the property accessor method  
508 \* @throws InvocationTargetException if the property accessor method  
509 \* throws an exception  
510 \*  
511 \* @see LocaleBeanUtilsBean#setProperty(Object, String, Object)  
512 \*/  
513 public static void setProperty(final Object bean, final String name, final Object value)  
514 throws IllegalAccessException, InvocationTargetException {  
515  
516 LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().setProperty(bean, name, value);  
517 }  
518  
519 /\*\*  
520 \* <p>Set the specified locale-sensitive property value, performing type  
521 \* conversions as required to conform to the type of the destination  
522 \* property using the specified conversion pattern.</p>  
523 \*  
524 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
525 \*  
526 \* @param bean Bean on which setting is to be performed  
527 \* @param name Property name (can be nested/indexed/mapped/combo)  
528 \* @param value Value to be set  
529 \* @param pattern The conversion pattern  
530 \*  
531 \* @throws IllegalAccessException if the caller does not have  
532 \* access to the property accessor method  
533 \* @throws InvocationTargetException if the property accessor method  
534 \* throws an exception  
535 \*  
536 \* @see LocaleBeanUtilsBean#setProperty(Object, String, Object, String)  
537 \*/  
538 public static void setProperty(final Object bean, final String name, final Object value, final String pattern)  
539 throws IllegalAccessException, InvocationTargetException {  
540  
541 LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().setProperty(bean, name, value, pattern);  
542 }  
543  
544 /\*\*  
545 \* <p>Calculate the property type.</p>  
546 \*  
547 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
548 \*  
549 \* @param target The bean  
550 \* @param name The property name  
551 \* @param propName The Simple name of target property  
552 \* @return The property's type  
553 \*  
554 \* @throws IllegalAccessException if the caller does not have  
555 \* access to the property accessor method  
556 \* @throws InvocationTargetException if the property accessor method  
557 \* throws an exception  
558 \*  
559 \* @see LocaleBeanUtilsBean#definePropertyType(Object, String, String)  
560 \*/  
561 protected static Class<?> definePropertyType(final Object target, final String name, final String propName)  
562 throws IllegalAccessException, InvocationTargetException {  
563  
564 return LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().definePropertyType(target, name, propName);  
565 }  
566  
567 /\*\*  
568 \* <p>Convert the specified value to the required type using the  
569 \* specified conversion pattern.</p>  
570 \*  
571 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
572 \*  
573 \* @param type The Java type of target property  
574 \* @param index The indexed subscript value (if any)  
575 \* @param value The value to be converted  
576 \* @param pattern The conversion pattern  
577 \* @return The converted value  
578 \* @see LocaleBeanUtilsBean#convert(Class, int, Object, String)  
579 \*/  
580 protected static Object convert(final Class<?> type, final int index, final Object value, final String pattern) {  
581  
582 return LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().convert(type, index, value, pattern);  
583 }  
584  
585 /\*\*  
586 \* <p>Convert the specified value to the required type.</p>  
587 \*  
588 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
589 \*  
590 \* @param type The Java type of target property  
591 \* @param index The indexed subscript value (if any)  
592 \* @param value The value to be converted  
593 \* @return The converted value  
594 \* @see LocaleBeanUtilsBean#convert(Class, int, Object)  
595 \*/  
596 protected static Object convert(final Class<?> type, final int index, final Object value) {  
597  
598 return LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().convert(type, index, value);  
599 }  
600  
601 /\*\*  
602 \* <p>Invoke the setter method.</p>  
603 \*  
604 \* <p>For more details see <code>LocaleBeanUtilsBean</code></p>  
605 \*  
606 \* @param target The bean  
607 \* @param propName The Simple name of target property  
608 \* @param key The Mapped key value (if any)  
609 \* @param index The indexed subscript value (if any)  
610 \* @param newValue The value to be set  
611 \*  
612 \* @throws IllegalAccessException if the caller does not have  
613 \* access to the property accessor method  
614 \* @throws InvocationTargetException if the property accessor method  
615 \* throws an exception  
616 \*  
617 \* @see LocaleBeanUtilsBean#invokeSetter(Object, String, String, int, Object)  
618 \*/  
619 protected static void invokeSetter(final Object target, final String propName, final String key, final int index, final Object newValue)  
620 throws IllegalAccessException, InvocationTargetException {  
621  
622 LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().invokeSetter(target, propName, key, index, newValue);  
623 }  
624  
625 /\*\*  
626 \* Resolve any nested expression to get the actual target bean.  
627 \*  
628 \* @deprecated moved into <code>LocaleBeanUtilsBean</code>  
629 \* @param bean The bean  
630 \* @param name The property name  
631 \* @return The property's descriptor  
632 \*  
633 \* @throws IllegalAccessException if the caller does not have  
634 \* access to the property accessor method  
635 \* @throws InvocationTargetException if the property accessor method  
636 \* throws an exception  
637 \*/  
638 @Deprecated  
639 protected static Descriptor calculate(final Object bean, final String name)  
640 throws IllegalAccessException, InvocationTargetException {  
641  
642 final org.apache.commons.beanutils.locale.LocaleBeanUtilsBean.Descriptor descriptor  
643 = LocaleBeanUtilsBean.getLocaleBeanUtilsInstance().calculate(bean, name);  
644 return new Descriptor(  
645 descriptor.getTarget(),  
646 descriptor.getName(),  
647 descriptor.getPropName(),  
648 descriptor.getKey(),  
649 descriptor.getIndex());  
650 }  
651  
652 /\*\* @deprecated moved into <code>LocaleBeanUtils</code> \*/  
653 @Deprecated  
654 protected static class Descriptor {  
655  
656 private int index = -1; // Indexed subscript value (if any)  
657 private String name;  
658 private String propName; // Simple name of target property  
659 private String key; // Mapped key value (if any)  
660 private Object target;  
661  
662 /\*\*  
663 \* Construct a descriptor instance for the target bean and property.  
664 \*  
665 \* @param target The target bean  
666 \* @param name The property name (includes indexed/mapped expr)  
667 \* @param propName The property name  
668 \* @param key The mapped property key (if any)  
669 \* @param index The indexed property index (if any)  
670 \*/  
671 public Descriptor(final Object target, final String name, final String propName, final String key, final int index) {  
672  
673 setTarget(target);  
674 setName(name);  
675 setPropName(propName);  
676 setKey(key);  
677 setIndex(index);  
678 }  
679  
680 /\*\*  
681 \* Return the target bean.  
682 \*  
683 \* @return The descriptors target bean  
684 \*/  
685 public Object getTarget() {  
686 return target;  
687 }  
688  
689 /\*\*  
690 \* Set the target bean.  
691 \*  
692 \* @param target The target bean  
693 \*/  
694 public void setTarget(final Object target) {  
695 this.target = target;  
696 }  
697  
698 /\*\*  
699 \* Return the mapped property key.  
700 \*  
701 \* @return the mapped property key (if any)  
702 \*/  
703 public String getKey() {  
704 return key;  
705 }  
706  
707 /\*\*  
708 \* Set the mapped property key.  
709 \*  
710 \* @param key The mapped property key (if any)  
711 \*/  
712 public void setKey(final String key) {  
713 this.key = key;  
714 }  
715  
716 /\*\*  
717 \* Return indexed property index.  
718 \*  
719 \* @return indexed property index (if any)  
720 \*/  
721 public int getIndex() {  
722 return index;  
723 }  
724  
725 /\*\*  
726 \* Set the indexed property index.  
727 \*  
728 \* @param index The indexed property index (if any)  
729 \*/  
730 public void setIndex(final int index) {  
731 this.index = index;  
732 }  
733  
734 /\*\*  
735 \* Return property name (includes indexed/mapped expr).  
736 \*  
737 \* @return The property name (includes indexed/mapped expr)  
738 \*/  
739 public String getName() {  
740 return name;  
741 }  
742  
743 /\*\*  
744 \* Set the property name (includes indexed/mapped expr).  
745 \*  
746 \* @param name The property name (includes indexed/mapped expr)  
747 \*/  
748 public void setName(final String name) {  
749 this.name = name;  
750 }  
751  
752 /\*\*  
753 \* Return the property name.  
754 \*  
755 \* @return The property name  
756 \*/  
757 public String getPropName() {  
758 return propName;  
759 }  
760  
761 /\*\*  
762 \* Set the property name.  
763 \*  
764 \* @param propName The property name  
765 \*/  
766 public void setPropName(final String propName) {  
767 this.propName = propName;  
768 }  
769 }  
770}  
771  
772