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  
018  
019package org.apache.commons.beanutils.converters;  
020  
021  
022import java.util.List;  
023import org.apache.commons.beanutils.ConversionException;  
024  
025  
026/\*\*  
027 \* Standard {@link org.apache.commons.beanutils.Converter} implementation that converts an incoming  
028 \* String into an array of String objects. On a conversion failure, returns  
029 \* a specified default value or throws a {@link ConversionException} depending  
030 \* on how this instance is constructed.  
031 \* <p>  
032 \* There is also some special handling where the input is of type int[].  
033 \* See method convert for more details.  
034 \*  
035 \* @version $Id$  
036 \* @since 1.4  
037 \* @deprecated Replaced by the new {@link ArrayConverter} implementation  
038 \*/  
039  
040@Deprecated  
041public final class StringArrayConverter extends AbstractArrayConverter {  
042  
043  
044 // ----------------------------------------------------------- Constructors  
045  
046  
047 /\*\*  
048 \* Create a {@link org.apache.commons.beanutils.Converter} that will throw  
049 \* a {@link ConversionException} if a conversion error occurs.  
050 \*/  
051 public StringArrayConverter() {  
052  
053 this.defaultValue = null;  
054 this.useDefault = false;  
055  
056 }  
057  
058  
059 /\*\*  
060 \* Create a {@link org.apache.commons.beanutils.Converter} that will return  
061 \* the specified default value if a conversion error occurs.  
062 \*  
063 \* @param defaultValue The default value to be returned  
064 \*/  
065 public StringArrayConverter(final Object defaultValue) {  
066  
067 this.defaultValue = defaultValue;  
068 this.useDefault = true;  
069  
070 }  
071  
072  
073 // ------------------------------------------------------- Static Variables  
074  
075  
076 /\*\*  
077 \* <p>Model object for type comparisons.</p>  
078 \*/  
079 private static final String[] MODEL = new String[0];  
080  
081 /\*\*  
082 \* <p> Model object for int arrays.</p>  
083 \*/  
084 private static final int[] INT\_MODEL = new int[0];  
085  
086  
087  
088 // --------------------------------------------------------- Public Methods  
089  
090  
091 /\*\*  
092 \* Convert the specified input object into an output object of the  
093 \* specified type.  
094 \* <p>  
095 \* If the value is already of type String[] then it is simply returned  
096 \* unaltered.  
097 \* <p>  
098 \* If the value is of type int[], then a String[] is returned where each  
099 \* element in the string array is the result of calling Integer.toString  
100 \* on the corresponding element of the int array. This was added as a  
101 \* result of bugzilla request #18297 though there is not complete  
102 \* agreement that this feature should have been added.  
103 \* <p>  
104 \* In all other cases, this method calls toString on the input object, then  
105 \* assumes the result is a comma-separated list of values. The values are  
106 \* split apart into the individual items and returned as the elements of an  
107 \* array. See class AbstractArrayConverter for the exact input formats  
108 \* supported.  
109 \*  
110 \* @param type is the data type to which this value should be converted.  
111 \* It is expected to be the class for type String[] (though this parameter  
112 \* is actually ignored by this method).  
113 \*  
114 \* @param value is the input value to be converted. If null then the  
115 \* default value is returned or an exception thrown if no default value  
116 \* exists.  
117 \* @return the converted value  
118 \*  
119 \* @throws ConversionException if conversion cannot be performed  
120 \* successfully, or the input is null and there is no default value set  
121 \* for this object.  
122 \*/  
123 @Override  
124 public Object convert(final Class type, final Object value) {  
125  
126 // Deal with a null value  
127 if (value == null) {  
128 if (useDefault) {  
129 return (defaultValue);  
130 } else {  
131 throw new ConversionException("No value specified");  
132 }  
133 }  
134  
135 // Deal with the no-conversion-needed case  
136 if (MODEL.getClass() == value.getClass()) {  
137 return (value);  
138 }  
139  
140 // Deal with the input value as an int array  
141 if (INT\_MODEL.getClass() == value.getClass())  
142 {  
143 final int[] values = (int[]) value;  
144 final String[] results = new String[values.length];  
145 for (int i = 0; i < values.length; i++)  
146 {  
147 results[i] = Integer.toString(values[i]);  
148 }  
149  
150 return (results);  
151 }  
152  
153 // Parse the input value as a String into elements  
154 // and convert to the appropriate type  
155 try {  
156 final List list = parseElements(value.toString());  
157 final String[] results = new String[list.size()];  
158 for (int i = 0; i < results.length; i++) {  
159 results[i] = (String) list.get(i);  
160 }  
161 return (results);  
162 } catch (final Exception e) {  
163 if (useDefault) {  
164 return (defaultValue);  
165 } else {  
166 throw new ConversionException(value.toString(), e);  
167 }  
168 }  
169 }  
170  
171}