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 \* <p>Standard {@link org.apache.commons.beanutils.Converter} implementation that converts an incoming  
028 \* String into a primitive array of float. On a conversion failure, returns  
029 \* a specified default value or throws a {@link ConversionException} depending  
030 \* on how this instance is constructed.</p>  
031 \*  
032 \* @version $Id$  
033 \* @since 1.4  
034 \* @deprecated Replaced by the new {@link ArrayConverter} implementation  
035 \*/  
036  
037@Deprecated  
038public final class FloatArrayConverter extends AbstractArrayConverter {  
039  
040  
041 // ----------------------------------------------------------- Constructors  
042  
043  
044 /\*\*  
045 \* Create a {@link org.apache.commons.beanutils.Converter} that will throw  
046 \* a {@link ConversionException} if a conversion error occurs.  
047 \*/  
048 public FloatArrayConverter() {  
049  
050 this.defaultValue = null;  
051 this.useDefault = false;  
052  
053 }  
054  
055  
056 /\*\*  
057 \* Create a {@link org.apache.commons.beanutils.Converter} that will return  
058 \* the specified default value if a conversion error occurs.  
059 \*  
060 \* @param defaultValue The default value to be returned  
061 \*/  
062 public FloatArrayConverter(final Object defaultValue) {  
063  
064 this.defaultValue = defaultValue;  
065 this.useDefault = true;  
066  
067 }  
068  
069  
070 // ------------------------------------------------------- Static Variables  
071  
072  
073 /\*\*  
074 \* <p>Model object for type comparisons.</p>  
075 \*/  
076 private static final float[] MODEL = new float[0];  
077  
078  
079 // --------------------------------------------------------- Public Methods  
080  
081  
082 /\*\*  
083 \* Convert the specified input object into an output object of the  
084 \* specified type.  
085 \*  
086 \* @param type Data type to which this value should be converted  
087 \* @param value The input value to be converted  
088 \* @return the converted value  
089 \*  
090 \* @throws ConversionException if conversion cannot be performed  
091 \* successfully  
092 \*/  
093 @Override  
094 public Object convert(final Class type, final Object value) {  
095  
096 // Deal with a null value  
097 if (value == null) {  
098 if (useDefault) {  
099 return (defaultValue);  
100 } else {  
101 throw new ConversionException("No value specified");  
102 }  
103 }  
104  
105 // Deal with the no-conversion-needed case  
106 if (MODEL.getClass() == value.getClass()) {  
107 return (value);  
108 }  
109  
110 // Deal with input value as a String array  
111 if (strings.getClass() == value.getClass()) {  
112 try {  
113 final String[] values = (String[]) value;  
114 final float[] results = new float[values.length];  
115 for (int i = 0; i < values.length; i++) {  
116 results[i] = Float.parseFloat(values[i]);  
117 }  
118 return (results);  
119 } catch (final Exception e) {  
120 if (useDefault) {  
121 return (defaultValue);  
122 } else {  
123 throw new ConversionException(value.toString(), e);  
124 }  
125 }  
126 }  
127  
128 // Parse the input value as a String into elements  
129 // and convert to the appropriate type  
130 try {  
131 final List list = parseElements(value.toString());  
132 final float[] results = new float[list.size()];  
133 for (int i = 0; i < results.length; i++) {  
134 results[i] = Float.parseFloat((String) list.get(i));  
135 }  
136 return (results);  
137 } catch (final Exception e) {  
138 if (useDefault) {  
139 return (defaultValue);  
140 } else {  
141 throw new ConversionException(value.toString(), e);  
142 }  
143 }  
144  
145 }  
146  
147  
148}