

PrefixFileFilter.java - SuffixFileFilter.java

```

1/*
2 * Licensed to the Apache Software Foundation (ASF) under one or more
3 * contributor license agreements. See the NOTICE file distributed with
4 * this work for additional information regarding copyright ownership.
5 * The ASF licenses this file to You under the Apache License, Version 2.0
6 * (the "License"); you may not use this file except in compliance with
7 * the License. You may obtain a copy of the License at
8 *
9 *     http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17package org.apache.commons.io.filefilter;
18
19import java.io.File;
20import java.io.Serializable;
21import java.util.List;
22
23import org.apache.commons.io.IOCase;
24
25/**
26 * Filters filenames for a certain prefix.
27 *
28 * <p>
29 * For example, to print all files and directories in the
30 * current directory whose name starts with <code>Test</code>:
31 *
32 * <pre>
33 * File dir = new File(".");
34 * String[] files = dir.list( new PrefixFileFilter("Test") );
35 * for (int i = 0; i < files.length; i++) {
36 *     System.out.println(files[i]);
37 * }
38 * </pre>
39 *
40 * @since Commons IO 1.0
41 * @version $Revision: 606381 $ $Date: 2007-12-22 02:03:16 +0000 (Sat, 22 Dec 2007) $
42 * @author Stephen Colebourne
43 * @author Federico Barbieri
44 * @author Serge Knystautas
45 * @author Peter Donald
46 */
47public class PrefixFileFilter extends AbstractFileFilter implements Serializable {
48
49     /** The filename prefixes to search for */
50     private final String[] prefixes;
51
52     /** Whether the comparison is case sensitive. */
53     private final IOCase caseSensitivity;
54

```

```

1/*
2 * Licensed to the Apache Software Foundation (ASF) under one or more
3 * contributor license agreements. See the NOTICE file distributed with
4 * this work for additional information regarding copyright ownership.
5 * The ASF licenses this file to You under the Apache License, Version 2.0
6 * (the "License"); you may not use this file except in compliance with
7 * the License. You may obtain a copy of the License at
8 *
9 *     http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17package org.apache.commons.io.filefilter;
18
19import java.io.File;
20import java.io.Serializable;
21import java.util.List;
22
23import org.apache.commons.io.IOCase;
24
25/**
26 * Filters files based on the suffix (what the filename ends with).
27 * This is used in retrieving all the files of a particular type.
28 *
29 * <p>
30 * For example, to retrieve and print all <code>*.java</code> files
31 * in the current directory:
32 *
33 * <pre>
34 * File dir = new File(".");
35 * String[] files = dir.list( new SuffixFileFilter(".java") );
36 * for (int i = 0; i < files.length; i++) {
37 *     System.out.println(files[i]);
38 * }
39 * </pre>
40 *
41 * @since Commons IO 1.0
42 * @version $Revision: 606381 $ $Date: 2007-12-22 02:03:16 +0000 (Sat, 22 Dec 2007) $
43 * @author Stephen Colebourne
44 * @author Federico Barbieri
45 * @author Serge Knystautas
46 * @author Peter Donald
47 */
48public class SuffixFileFilter extends AbstractFileFilter implements Serializable {
49
50     /** The filename suffixes to search for */
51     private final String[] suffixes;
52
53     /** Whether the comparison is case sensitive. */
54     private final IOCase caseSensitivity;
55

```

PrefixFileFilter.java - SuffixFileFilter.java

```

55  /**
56  * Constructs a new Prefix file filter for a single prefix.
57  *
58  * @param prefix the prefix to allow, must not be null
59  * @throws IllegalArgumentException if the prefix is null
60  */
61  public PrefixFileFilter(String prefix) {
62      this(prefix, IOCase.SENSITIVE);
63  }
64
65  /**
66  * Constructs a new Prefix file filter for a single prefix
67  * specifying case-sensitivity.
68  *
69  * @param prefix the prefix to allow, must not be null
70  * @param caseSensitivity how to handle case sensitivity, null means
case-sensitive
71  * @throws IllegalArgumentException if the prefix is null
72  * @since Commons IO 1.4
73  */
74  public PrefixFileFilter(String prefix, IOCase caseSensitivity) {
75      if (prefix == null) {
76          throw new IllegalArgumentException("The prefix must not be null");
77      }
78      this.prefixes = new String[] {prefix};
79      this.caseSensitivity = (caseSensitivity == null ? IOCase.SENSITIVE :
caseSensitivity);
80  }
81
82  /**
83  * Constructs a new Prefix file filter for any of an array of prefixes.
84  * <p>
85  * The array is not cloned, so could be changed after constructing the
86  * instance. This would be inadvisable however.
87  *
88  * @param prefixes the prefixes to allow, must not be null
89  * @throws IllegalArgumentException if the prefix array is null
90  */
91  public PrefixFileFilter(String[] prefixes) {
92      this(prefixes, IOCase.SENSITIVE);
93  }
94
95  /**
96  * Constructs a new Prefix file filter for any of an array of prefixes
97  * specifying case-sensitivity.
98  * <p>
99  * The array is not cloned, so could be changed after constructing the
100  * instance. This would be inadvisable however.
101  *
102  * @param prefixes the prefixes to allow, must not be null
103  * @param caseSensitivity how to handle case sensitivity, null means
case-sensitive
104  * @throws IllegalArgumentException if the prefix is null
105  * @since Commons IO 1.4
106  */
107  public PrefixFileFilter(String[] prefixes, IOCase caseSensitivity) {

```

```

56  /**
57  * Constructs a new Suffix file filter for a single extension.
58  *
59  * @param suffix the suffix to allow, must not be null
60  * @throws IllegalArgumentException if the suffix is null
61  */
62  public SuffixFileFilter(String suffix) {
63      this(suffix, IOCase.SENSITIVE);
64  }
65
66  /**
67  * Constructs a new Suffix file filter for a single extension
68  * specifying case-sensitivity.
69  *
70  * @param suffix the suffix to allow, must not be null
71  * @param caseSensitivity how to handle case sensitivity, null means
case-sensitive
72  * @throws IllegalArgumentException if the suffix is null
73  * @since Commons IO 1.4
74  */
75  public SuffixFileFilter(String suffix, IOCase caseSensitivity) {
76      if (suffix == null) {
77          throw new IllegalArgumentException("The suffix must not be null");
78      }
79      this.suffixes = new String[] {suffix};
80      this.caseSensitivity = (caseSensitivity == null ? IOCase.SENSITIVE :
caseSensitivity);
81  }
82
83  /**
84  * Constructs a new Suffix file filter for an array of suffixes.
85  * <p>
86  * The array is not cloned, so could be changed after constructing the
87  * instance. This would be inadvisable however.
88  *
89  * @param suffixes the suffixes to allow, must not be null
90  * @throws IllegalArgumentException if the suffix array is null
91  */
92  public SuffixFileFilter(String[] suffixes) {
93      this(suffixes, IOCase.SENSITIVE);
94  }
95
96  /**
97  * Constructs a new Suffix file filter for an array of suffixes
98  * specifying case-sensitivity.
99  * <p>
100  * The array is not cloned, so could be changed after constructing the
101  * instance. This would be inadvisable however.
102  *
103  * @param suffixes the suffixes to allow, must not be null
104  * @param caseSensitivity how to handle case sensitivity, null means
case-sensitive
105  * @throws IllegalArgumentException if the suffix array is null
106  * @since Commons IO 1.4
107  */
108  public SuffixFileFilter(String[] suffixes, IOCase caseSensitivity) {

```

PrefixFileFilter.java - SuffixFileFilter.java

```

108     if (prefixes == null) {
109         throw new IllegalArgumentException("The array of prefixes must not be
110         null");
111     }
112     this.prefixes = prefixes;
113     this.caseSensitivity = (caseSensitivity == null ? IOCase.SENSITIVE :
114     caseSensitivity);
115 }
116 /**
117  * Constructs a new Prefix file filter for a list of prefixes.
118  *
119  * @param prefixes the prefixes to allow, must not be null
120  * @throws IllegalArgumentException if the prefix list is null
121  * @throws ClassCastException if the list does not contain Strings
122  */
123 public PrefixFileFilter(List prefixes) {
124     this(prefixes, IOCase.SENSITIVE);
125 }
126 /**
127  * Constructs a new Prefix file filter for a list of prefixes
128  * specifying case-sensitivity.
129  *
130  * @param prefixes the prefixes to allow, must not be null
131  * @param caseSensitivity how to handle case sensitivity, null means
132  * case-sensitive
133  * @throws IllegalArgumentException if the prefix list is null
134  * @throws ClassCastException if the list does not contain Strings
135  * @since Commons IO 1.4
136 */
137 public PrefixFileFilter(List prefixes, IOCase caseSensitivity) {
138     if (prefixes == null) {
139         throw new IllegalArgumentException("The list of prefixes must not be
140         null");
141     }
142     this.prefixes = (String[]) prefixes.toArray(new String[prefixes.size()]);
143     this.caseSensitivity = (caseSensitivity == null ? IOCase.SENSITIVE :
144     caseSensitivity);
145 }
146 /**
147  * Checks to see if the filename starts with the prefix.
148  *
149  * @param file the File to check
150  * @return true if the filename starts with one of our prefixes
151  */
152 public boolean accept(File file) {
153     String name = file.getName();
154     for (int i = 0; i < this.prefixes.length; i++) {
155         if (caseSensitivity.checkStartsWith(name, prefixes[i])) {
156             return true;
157         }
158     }
159     return false;
160 }

```

```

109     if (suffixes == null) {
110         throw new IllegalArgumentException("The array of suffixes must not be
111         null");
112     }
113     this.suffixes = suffixes;
114     this.caseSensitivity = (caseSensitivity == null ? IOCase.SENSITIVE :
115     caseSensitivity);
116 }
117 /**
118  * Constructs a new Suffix file filter for a list of suffixes.
119  *
120  * @param suffixes the suffixes to allow, must not be null
121  * @throws IllegalArgumentException if the suffix list is null
122  * @throws ClassCastException if the list does not contain Strings
123  */
124 public SuffixFileFilter(List suffixes) {
125     this(suffixes, IOCase.SENSITIVE);
126 }
127 /**
128  * Constructs a new Suffix file filter for a list of suffixes
129  * specifying case-sensitivity.
130  *
131  * @param suffixes the suffixes to allow, must not be null
132  * @param caseSensitivity how to handle case sensitivity, null means
133  * case-sensitive
134  * @throws IllegalArgumentException if the suffix list is null
135  * @throws ClassCastException if the list does not contain Strings
136  * @since Commons IO 1.4
137 */
138 public SuffixFileFilter(List suffixes, IOCase caseSensitivity) {
139     if (suffixes == null) {
140         throw new IllegalArgumentException("The list of suffixes must not be
141         null");
142     }
143     this.suffixes = (String[]) suffixes.toArray(new String[suffixes.size()]);
144     this.caseSensitivity = (caseSensitivity == null ? IOCase.SENSITIVE :
145     caseSensitivity);
146 }
147 /**
148  * Checks to see if the filename ends with the suffix.
149  *
150  * @param file the File to check
151  * @return true if the filename ends with one of our suffixes
152  */
153 public boolean accept(File file) {
154     String name = file.getName();
155     for (int i = 0; i < this.suffixes.length; i++) {
156         if (caseSensitivity.checkEndsWith(name, suffixes[i])) {
157             return true;
158         }
159     }
160     return false;
161 }

```

PrefixFileFilter.java - SuffixFileFilter.java

```

159
160 /**
161  * Checks to see if the filename starts with the prefix.
162  *
163  * @param file the File directory
164  * @param name the filename
165  * @return true if the filename starts with one of our prefixes
166  */
167 public boolean accept(File file, String name) {
168     for (int i = 0; i < prefixes.length; i++) {
169         if (caseSensitivity.checkStartsWith(name, prefixes[i])) {
170             return true;
171         }
172     }
173     return false;
174 }
175
176 /**
177  * Provide a String representaion of this file filter.
178  *
179  * @return a String representaion
180  */
181 public String toString() {
182     StringBuffer buffer = new StringBuffer();
183     buffer.append(super.toString());
184     buffer.append("(");
185     if (prefixes != null) {
186         for (int i = 0; i < prefixes.length; i++) {
187             if (i > 0) {
188                 buffer.append(", ");
189             }
190             buffer.append(prefixes[i]);
191         }
192     }
193     buffer.append(")");
194     return buffer.toString();
195 }
196
197 }
198

```

```

160
161 /**
162  * Checks to see if the filename ends with the suffix.
163  *
164  * @param file the File directory
165  * @param name the filename
166  * @return true if the filename ends with one of our suffixes
167  */
168 public boolean accept(File file, String name) {
169     for (int i = 0; i < this.suffixes.length; i++) {
170         if (caseSensitivity.checkEndsWith(name, suffixes[i])) {
171             return true;
172         }
173     }
174     return false;
175 }
176
177 /**
178  * Provide a String representaion of this file filter.
179  *
180  * @return a String representaion
181  */
182 public String toString() {
183     StringBuffer buffer = new StringBuffer();
184     buffer.append(super.toString());
185     buffer.append("(");
186     if (suffixes != null) {
187         for (int i = 0; i < suffixes.length; i++) {
188             if (i > 0) {
189                 buffer.append(", ");
190             }
191             buffer.append(suffixes[i]);
192         }
193     }
194     buffer.append(")");
195     return buffer.toString();
196 }
197
198 }
199

```