Fully documented source code for the ReporterIonAgent

```
1  package peptizer.agents.custom;
2
3  import be.proteomics.mascotdatfile.util.interfaces.Spectrum;
4  import be.proteomics.mascotdatfile.util.mascot.Peak;
5  import be.proteomics.mat.MatConfig;
6  import be.proteomics.mat.interfaces.Agent;
7  import be.proteomics.mat.util.AgentReport;
8  import be.proteomics.mat.util.PeptideIdentification;
9  import be.proteomics.mat.util.enumerator.AgentVote;
10 import be.proteomics.mat.util.fileio.MatLogger;
11
12 import java.math.BigDecimal;
13 import java.util.Properties;
14 /**
15  * Created by IntelliJ IDEA.
16  * User: kenny
17  * Date: 18-jun-2008
18  * Time: 15:30:28
19  */
20
21 /**
22  * Class description:
23  * ------------------
24  * This class was developed to inspect for deviating reporter ion intensities.
25  */
26 public class ReporterIonAgent extends Agent {
27
28     /**
29     * PARAMETERS
30     * ----------
31     * String identifiers for the parameters in the agent.xml configuration file.
32     */
33
34     // Mass over charge parameter for the first reporter ion.
35     public static final String MASS_1 = "reporter_mz_1";
36
37     // Mass over charge parameter for the second reporter ion.
38     public static final String MASS_2 = "reporter_mz_2";
39
40     // Fold ratio parameter between the two reporter ions you consider as deviating.
41     public static final String RATIO = "ratio";
42
43     // Error tolerance for matching the expected reporter
44     // ion mass over charge values to a fragmention in the MS/MS spectrum.
45     public static final String ERROR = "error";
46
47     /**
48     * CONSTRUCTOR
49     * -----------
50     * Construct a new instance of the ReporterIonAgent.
51     */
52     public ReporterIonAgent() {
53       // 1. Calls the constructor of the Agent superclass.
54       super();
55       // 2. Gets the properties for this Agent.
56       // A singleton MatConfig object reads the agent.xml configuration file upon starting Peptizer.
57       // The Properties of each Agent can be then be retrieved by the unique identifier of an Agent.
58       Properties prop = MatConfig.getInstance().getAgentProperties(this.getUniqueID());
59
60       try {
```

Fully documented source code for the ReporterIonAgent

```
61          // 2a. Sets common properties shared among all Agents.
62          super.setName(prop.getProperty("name"));
63          super.setActive(Boolean.valueOf(prop.getProperty("active")));
64          super.setVeto(Boolean.valueOf(prop.getProperty("veto")));
65
66          // 2b. Sets specific properties for this ReporterIonAgent.
67          // Masses of the two reporter ions, the threshold ratio and the mass error tolerance.
68          this.iProperties.put(MASS_1, prop.getProperty(MASS_1));
69          this.iProperties.put(MASS_2, prop.getProperty(MASS_2));
70          this.iProperties.put(RATIO, prop.getProperty(RATIO));
71          this.iProperties.put(ERROR, prop.getProperty(ERROR));
72        } catch (NullPointerException npe) {
73          // Note that an exception is thrown when one of the parameters is missing!
74          MatLogger.logExceptionalGUIMessage("Missing Parameter!!",
75                "Parameters " + MASS_1 + ", " + MASS_2 + ", " + RATIO + "and " + ERROR +
76                    " are required! for Agent \"" + this.getName() + "\" !!\nExiting..");
77          System.exit(0);
78        }
79    }
80
81    /**
82    * INSPECTION
83    * ----------
84    * The inspection is the core of an Agent since this logic leads to the Agent's vote.
85
86    * This method returns an array of AgentVote objects, reflecting this Agent's idea
87    * whether to select or not to select the peptide hypothesis.
88
89    * All Agent Implementations must also create and store AgentReport for each peptide hypothesis.
90    *
91    * @param aPeptideIdentification PeptideIdentification that has to be inspected.
92    * @return AgentVote[] as a vote upon inspection for each the confident peptide hypothesises.
93    *       AgentVotes[0] gives the inspection result on PeptideHit 1
94    *       AgentVotes[1] gives the inspection result on PeptideHit 2
95    *       AgentVotes[n] gives the inspection result on PeptideHit n+1
96
97    *       Where the different AgentVotes can be:
98
99    *       a vote approving the selection of the peptide hypothesis.
100   *       a vote indifferent to the selection.
101   *       a vote objecting to select the peptide hypothesis.
102   */
103   public AgentVote[] inspect(PeptideIdentification aPeptideIdentification) {
104
105     // A. PREPARING THE VARIABLES
106     //****************************
107
108     // 1. The reporter ion masses.
109     double lReporterMass_1 = Double.parseDouble((String) (this.iProperties.get(MASS_1)));
110     double lReporterMass_2 = Double.parseDouble((String) (this.iProperties.get(MASS_2)));
111
112     // 2. The fold ratio threshold.
113     double lRatio = Double.parseDouble((String) (this.iProperties.get(RATIO)));
114
115     // 3. The error tolerance.
116     double lError = Double.parseDouble((String) (this.iProperties.get(ERROR)));
117
118     // 4. Reserves an array with AgentVotes for each confident peptide hypothesis.
119     AgentVote[] lAgentVotes =
120         new AgentVote[aPeptideIdentification.getNumberOfConfidentPeptideHits()];
```

Fully documented source code for the ReporterIonAgent

```
121
122         // Since this inspection is dependent on the MS/MS spectrum,
123         // it will result in the same vote for each peptide hypothesis.
124         // Therefore, a single inspection is reused for each peptide hypothesis.
125
126         // 5. Initiate an AgentReport serving as a report for this inspection.
127         iReport = new AgentReport(getUniqueID());
128
129         // 6. Local variable to store the result shown in the information table.
130         String lResultForTable = "";
131         // 7. Local variable to store the result written in the arff file.
132         String lResultForArff = "";
133
134         // 8. Local variables for matching reporter ion 1 in the MS/MS spectrum.
135         boolean lReporter_1_match = false;
136         double lReporter_1_intensity = 0;
137
138         // 9. Local variables for matching reporter ion 2 in the MS/MS spectrum.
139         boolean lReporter_2_match = false;
140         double lReporter_2_intensity = 0;
141
142         // B. THE ACTUAL INSPECTION
143         //**************************************
144
145         // 1. Gets the MS/MS spectrum from the PeptideIdentification object
146         // that was given as a parameter to the inspect() method.
147         Spectrum lSpectrum = aPeptideIdentification.getSpectrum();
148
149         // 2. Gets the peaklist from this MS/MS spectrum.
150         Peak[] lPeaks = lSpectrum.getPeakList();
151
152         // 3. Iterates over all peaks through a for loop.
153         for (int i = 0; i < lPeaks.length; i++) {
154             Peak lPeak = lPeaks[i];
155             double lDelta_1 = lPeak.getMZ() - lReporterMass_1;
156             double lDelta_2 = lPeak.getMZ() - lReporterMass_2;
157
158             // 3i) If absolute value of the mass diference of this fragmention and the expected mass
159             // of reporter ion 1 is less then the defined error tolerance.
160             // Then there is a match!
161             if (Math.abs(lDelta_1) < lError) {
162                 lReporter_1_match = true;
163                 lReporter_1_intensity = lPeak.getIntensity();
164             }
165
166             // 3ii) Same idea for reporter ion 2.
167             if (Math.abs(lDelta_2) < lError) {
168                 lReporter_2_match = true;
169                 lReporter_2_intensity = lPeak.getIntensity();
170             }
171
172             // 3iii) For performance reasons: if both peaks were matched then the for loop can be exited.
173             if (lReporter_1_match && lReporter_2_match) {
174                 break;
175             }
176         }
177
178         // 4. Checks the intensity ratio.
179         double lUpperBoundary;
180         double lLowerBoundary;
```

Fully documented source code for the ReporterIonAgent

```
181
182     // First an upper and a lower boundary must be defined for the Reporter Ions intensity ratio.
183     // If the experimental ratio between Reporter Ion 1 and Reporter Ion 2 is
184     //  more then 1.5 or less then 0.66, then the two samples deviate by a factor of 1.5.
185     //
186     // If the user defined the factor as larger then 1, the upper boundary for an
187     // deviating ratio is given by 1 diveded by that factor.
188     // The lower boundary for an deviating ratio is given by 1 multiplied by that factor.
189     // Example:
190     // If (Ratio=1.5)
191     // Then lower boundary = 1.5 and upper boundary = 0.66
192     if (lRatio > 1) {
193        lUpperBoundary = 1 * lRatio;
194        lLowerBoundary = 1 / lRatio;
195     } else {
196        // If the user defined the facotr as smaller then 1, then it is the other way round.
197        lUpperBoundary = 1 / lRatio;
198        lLowerBoundary = 1 * lRatio;
199     }
200
201     // 5. Local variable for the intesity ratio between Reporter Ion 1 and Reporter Ion 2.
202     double lExperimentalRatio;
203     // Local boolean for the upcomming function to store whether the ratio between the Reporter Ions
204     // deviates more then the expected ratio.
205     boolean lDeviatingRatio;
206
207     // 6. Checks the intensities of the reporter ions!
208
209     // 6i) If this condition is true, then one of the reporter ions was not found!
210     // These are not selected as these are probably unlabeled peptides.
211     if (!lReporter_1_match || !lReporter_2_match) {
212        lDeviatingRatio = false;
213        lExperimentalRatio = 0;
214        // 6ii) Else both the reporter ions were found. Lets inspect their ratio.
215     } else {
216        lExperimentalRatio = lReporter_1_intensity / lReporter_2_intensity;
217
218        // The ExperimentalRatio between reporter ion 1 an reporter ion 2
219        // is either less then the lower boundary,
220        // or either more then the upper boundar.
221        // In both cases, the reporter ion intesity is deviating for both samples:
222        if (lLowerBoundary > lExperimentalRatio || lUpperBoundary < lExperimentalRatio) {
223           // A. The Agent inspection resulted in deviating reporter ion intensities as
224           // their ratio was outside one of the boundaries.
225           lDeviatingRatio = true;
226        } else {
227           // B. Else the Agent inspection resulted in non deviating reporter ion intensities as
228           // their ratio was within the lower and upper boundary.
229           lDeviatingRatio = false;
230        }
231     }
232
233     // C. MAKING THE INSPECTION REPORTS AND COMMITTING THE VOTES
234     //*********************************************************
235
236     // 1. In all cases, store the experimental ratio between the
237     // two reporter ions as a value to display in the information table.
238
239     // A BigDecimal rounds a double at 2 decimals.
240     BigDecimal lRoundedExperimentalRatio = null;
```

Fully documented source code for the ReporterIonAgent

```
241         lRoundedExperimentalRatio = new BigDecimal(lExperimentalRatio).setScale(2, BigDecimal.ROUND
    _HALF_UP);
242         lResultForTable = lRoundedExperimentalRatio.toString();
243
244         // 2i. Deviating reporter ion intesity ratio, this Agent suggests to select the peptide hypothesis!
245         if (lDeviatingRatio) {
246           lResultForArff = "1";
247           for (int i = 0; i < lAgentVotes.length; i++) {
248             lAgentVotes[i] = AgentVote.POSITIVE_FOR_SELECTION;
249           }
250           // 2ii. Non Deviating reporter ion intensity ratio, this Agent is neutral to select the peptide hypothesis!
251         } else {
252           lResultForArff = "0";
253           for (int i = 0; i < lAgentVotes.length; i++) {
254             lAgentVotes[i] = AgentVote.NEUTRAL_FOR_SELECTION;
255           }
256         }
257
258         // 3. Creates an Agentreport for this inspection.
259         iReport.addReport(AgentReport.RK_RESULT, lAgentVotes[0]);
260         iReport.addReport(AgentReport.RK_TABLEDATA, lResultForTable);
261         iReport.addReport(AgentReport.RK_ARFF, lResultForArff);
262
263         // 4. Stores the report on the PeptideIdentification object.
264         for (int i = 0; i < lAgentVotes.length; i++) {
265           aPeptideIdentification.addAgentReport((i + 1), this.getUniqueID(), iReport);
266         }
267
268         // 5. Returns the AgentVotes in the end of the inspection.
269         return lAgentVotes;
270       }
271
272       /**
273       * Returns a description for the Agent.
274       * Note that html tags are used to stress properties.
275       * Use in tooltips and configuration settings.
276       * <p/>
277       * Fill in an agent description. Report on purpose and a minor on actual implementation.
278       *
279       * @return String description of the ReporterIonAgent.
280       */
281       public String getDescription() {
282         return "<html>Inspects for the abberant reporter ion intensities." +
283             "<b>Selects when two reporter ions ( " + this.iProperties.get(MASS_1) +
284             " , " + this.iProperties.get(MASS_2) + ") have a more then " + this.iProperties.get(RATIO) +
285             " fold intesity ratio.";
286       }
287     }
```