```
 1  using System;
 2  using System.Collections;
 3  using System.Collections.Generic;
 4  using Impinj.OctaneSdk;
 5
 6
 7
 8  namespace TG2_RFID
 9  {
10      public class Cardholder
11      {
12
13          public static int RSSI_HIGHPASSFILTER = -60;
14          public static double DOPPLER_FILTER = 0.5;
15
16          /// <summary>
17          /// Holds the cardholder's name.
18          /// </summary>
19          protected string name;
20
21          /// <summary>
22          /// Holds the thermal capacity for this cardholder.
23          /// </summary>
24          protected int thermalCapacity;
25
26          /// <summary>
27          /// Holds the tag epc.
28          /// </summary>
29          protected string tagEPC;
30
31          /// <summary>
32          /// Holds whether the cardholder was initialized.
33          /// </summary>
34          protected bool wasInitialized;
35
36          /// <summary>
37          /// Holds the last seen tag as a Tag.
38          /// </summary>
39          protected Tag lastSeenTag;
40
41          /// <summary>
42          /// Holds the first seen time for this cardholder.
43          /// </summary>
44          protected DateTime firstSeenTime;
45
46          /// <summary>
47          /// Holds the last seen time for this cardholder.
48          /// </summary>
49          protected DateTime lastSeenTime;
50
51          /// <summary>
52          /// Holds the last seen rssi power value.
53          /// </summary>
54          protected double lastSeenRSSI;
55
56          /// <summary>
```

```
57          /// Holds the current ambient in which this cardholder is according to
                the default criteria.
58          /// </summary>
59          protected Ambient currentAmbient;
60
61          /// <summary>
62          /// Holds the current ambient in which this cardholder is according to
                the last RSSI peak time criteria.
63          /// </summary>
64          protected Ambient ambPeakTimes;
65
66          /// <summary>
67          /// Holds the current ambient in which this cardholder is according to
                the last RSSI peak time and magnitude criteria.
68          /// </summary>
69          protected Ambient ambPeakTimeAndMag;
70
71          /// <summary>
72          /// Holds the current ambient in which this cardholder is according to
                the last RSSI value criteria.
73          /// </summary>
74          protected Ambient ambLastRSSI;
75
76          /// <summary>
77          /// Holds the current ambient in which this cardholder is according to
                the last RSSI mean criteria.
78          /// </summary>
79          protected Ambient ambRSSIMean;
80
81          /// <summary>
82          /// Holds the current ambient in which this cardholder is according to
                the last RSSI median criteria.
83          /// </summary>
84          protected Ambient ambRSSIMedian;
85
86          /// <summary>
87          /// Holds the current ambient in which this cardholder is according to
                the Doppler Effect criteria.
88          /// </summary>
89          protected Ambient ambDopplerEffect;
90
91          /// <summary>
92          /// Holds the current ambient in which this cardholder is according to
                the Doppler Effect and RSSI criteria.
93          /// </summary>
94          protected Ambient ambDopplerAndRSSI;
95
96          /// <summary>
97          /// Holds a map of curves per antenna in which this cardholder was
              seen
98          /// holding the history of RSSI power per time.
99          /// </summary>
100         public Dictionary<Tuple<string, ushort>, Curve>
              curvesPowerReadingsDictionary;
101
102         /// <summary>
```

```
103         /// Holds a map of curves per antenna in which this cardholder was
              seen
104         /// holding the history of doppler frequency per time.
105         /// </summary>
106         public Dictionary<Tuple<string, ushort>, Curve>
              curvesDoplerFrequencyReadingsDictionary;
107
108         /// <summary>
109         /// Initializes a new instance of the <see
              cref="T:TG2_RFID.Cardholder"/> class.
110         /// </summary>
111         public Cardholder()
112         {
113             name = "DefaultName";
114             thermalCapacity = 1000;
115             wasInitialized = false;
116             curvesPowerReadingsDictionary = new Dictionary<Tuple<string,
                  ushort>, Curve>();
117             curvesDoplerFrequencyReadingsDictionary = new
                  Dictionary<Tuple<string, ushort>, Curve>();
118             ambDopplerAndRSSI = (Project.GetAmbientInstance(0));
119             ambDopplerEffect = (Project.GetAmbientInstance(0));
120             ambLastRSSI = (Project.GetAmbientInstance(0));
121             ambPeakTimeAndMag = (Project.GetAmbientInstance(0));
122             ambPeakTimes = (Project.GetAmbientInstance(0));
123             ambRSSIMean = (Project.GetAmbientInstance(0));
124             ambRSSIMedian = (Project.GetAmbientInstance(0));
125             currentAmbient = (Project.GetAmbientInstance(0));
126
127         }
128
129         /// <summary>
130         /// Initializes a new instance of the <see
              cref="T:TG2_RFID.Cardholder"/> class.
131         /// </summary>
132         /// <param name="personName">Person name.</param>
133         public Cardholder(string personName)
134         {
135             name = personName;
136             thermalCapacity = 1000;
137             wasInitialized = false;
138             curvesPowerReadingsDictionary = new Dictionary<Tuple<string,
                  ushort>, Curve>();
139             curvesDoplerFrequencyReadingsDictionary = new
                  Dictionary<Tuple<string, ushort>, Curve>();
140             ambDopplerAndRSSI = (Project.GetAmbientInstance(0));
141             ambDopplerEffect = (Project.GetAmbientInstance(0));
142             ambLastRSSI = (Project.GetAmbientInstance(0));
143             ambPeakTimeAndMag = (Project.GetAmbientInstance(0));
144             ambPeakTimes = (Project.GetAmbientInstance(0));
145             ambRSSIMean = (Project.GetAmbientInstance(0));
146             ambRSSIMedian = (Project.GetAmbientInstance(0));
147             currentAmbient = (Project.GetAmbientInstance(0));
148         }
149
150         /// <summary>
```

```
151         /// Initializes a new instance of the <see
              cref="T:TG2_RFID.Cardholder"/> class.
152         /// </summary>
153         /// <param name="personName">Person name.</param>
154         /// <param name="personThermalCapacity">Person thermal capacity.</
              param>
155         public Cardholder(string personName, int personThermalCapacity)
156         {
157             name = personName;
158             thermalCapacity = personThermalCapacity;
159             wasInitialized = false;
160             curvesPowerReadingsDictionary = new Dictionary<Tuple<string,
                  ushort>, Curve>();
161             curvesDoplerFrequencyReadingsDictionary = new
                  Dictionary<Tuple<string, ushort>, Curve>();
162             ambDopplerAndRSSI = (Project.GetAmbientInstance(0));
163             ambDopplerEffect = (Project.GetAmbientInstance(0));
164             ambLastRSSI = (Project.GetAmbientInstance(0));
165             ambPeakTimeAndMag = (Project.GetAmbientInstance(0));
166             ambPeakTimes = (Project.GetAmbientInstance(0));
167             ambRSSIMean = (Project.GetAmbientInstance(0));
168             ambRSSIMedian = (Project.GetAmbientInstance(0));
169             currentAmbient = (Project.GetAmbientInstance(0));
170         }
171
172         /// <summary>
173         /// Initializes a new instance of the <see
              cref="T:TG2_RFID.Cardholder"/> class.
174         /// </summary>
175         /// <param name="personName">Person name.</param>
176         /// <param name="personTagEPC">Person tag epc.</param>
177         public Cardholder(string personName, string personTagEPC)
178         {
179             name = personName;
180             thermalCapacity = 1000;
181             tagEPC = personTagEPC;
182             wasInitialized = false;
183             curvesPowerReadingsDictionary = new Dictionary<Tuple<string,
                  ushort>, Curve>();
184             curvesDoplerFrequencyReadingsDictionary = new
                  Dictionary<Tuple<string, ushort>, Curve>();
185         }
186
187         /// <summary>
188         /// Getter for the tag EPC.
189         /// </summary>
190         public string GetCardholderEPC()
191         {
192             return tagEPC;
193         }
194
195         /// <summary>
196         /// Setter for the tag EPC.
197         /// </summary>
198         /// <param name="newTagEPC">New tag epc.</param>
199         public void SetCardholderEPC(string newTagEPC)
```

```
200         {
201             tagEPC = newTagEPC;
202         }
203
204         /// <summary>
205         /// Getter for a RSSI power curve given an antenna.
206         /// </summary>
207         /// <returns>The power curve.</returns>
208         /// <param name="antenna">Antenna.</param>
209         public Curve GetPowerCurve(Tuple<string, ushort> antenna)
210         {
211             if (!curvesPowerReadingsDictionary.ContainsKey(antenna))
212             {
213                 curvesPowerReadingsDictionary.Add(antenna, new Curve());
214             }
215             curvesPowerReadingsDictionary.TryGetValue(antenna, out Curve
                retCurve);
216             return retCurve;
217         }
218
219         /// <summary>
220         /// Getter for the dopple effect curve given an antenna.
221         /// </summary>
222         /// <returns>The doppler effect curve.</returns>
223         /// <param name="antenna">Antenna.</param>
224         public Curve GetDopplerEffectCurve(Tuple<string, ushort> antenna)
225         {
226             if (!curvesDoplerFrequencyReadingsDictionary.ContainsKey(antenna))
227             {
228                 curvesDoplerFrequencyReadingsDictionary.Add(antenna, new Curve
                    ());
229             }
230             curvesDoplerFrequencyReadingsDictionary.TryGetValue(antenna, out
                Curve retCurve);
231             return retCurve;
232         }
233
234         /// <summary>
235         /// Getter for the tag EPC.
236         /// </summary>
237         /// <returns>The tag epc.</returns>
238         public string GetTagEPC()
239         {
240             return tagEPC;
241         }
242
243         /// <summary>
244         /// Getter for the person's name.
245         /// </summary>
246         public string GetName()
247         {
248             return name;
249         }
250
251         /// <summary>
252         /// TODO
```

```
253         /// TODO Reset buffer at 4am due to overflow problem.
254         /// </summary>
255         public void ReadingCardholderTag(Tag tag, string senderName)
256         {
257             lastSeenTag = tag;
258             lastSeenTime = DateTime.Now;
259             lastSeenRSSI = tag.PeakRssiInDbm;
260 
261 
262             if (!wasInitialized)
263             {
264                 firstSeenTime = lastSeenTime;
265                 wasInitialized = true;
266             }
267 
268             var diffTime = lastSeenTime - firstSeenTime;
269             double readingTime = (double)(diffTime.TotalMilliseconds);
270 
271             var tupleAntenna = Tuple.Create<string, ushort>(senderName,
                  tag.AntennaPortNumber);
272 
273             if (!curvesPowerReadingsDictionary.ContainsKey(tupleAntenna))
274             {
275                 curvesPowerReadingsDictionary.Add(tupleAntenna, new Curve());
276             }
277             var powerCurve = curvesPowerReadingsDictionary[tupleAntenna];
278             if (tag.PeakRssiInDbm > RSSI_HIGHPASSFILTER)
279             {
280                 powerCurve.AddPointWithAvgFilter(readingTime,
                      tag.PeakRssiInDbm);
281             }
282             // foreach (var antenna in curvesPowerReadingsDictionary.Keys)
283             //{
284             //    cnt++;
285             //    var powerCurve = curvesPowerReadingsDictionary[antenna];
286             //    if (tupleAntenna.Equals(antenna))
287             //    {
288             //        powerCurve.AddPointWithAvgFilter(readingTime,
                  tag.PeakRssiInDbm);
289             //    }
290             //    else
291             //    {
292             //        powerCurve.AddPointWithAvgFilter(readingTime, 0.0);
293             //    }
294             //}
295 
296 
297             if (!curvesDoplerFrequencyReadingsDictionary.ContainsKey
                  (tupleAntenna))
298             {
299                 curvesDoplerFrequencyReadingsDictionary.Add(tupleAntenna, new
                      Curve());
300             }
301             try
302             {
303                 var dopplerCurve = curvesDoplerFrequencyReadingsDictionary
```

```
                    [tupleAntenna];
304                 if (Math.Abs(tag.RfDopplerFrequency) > DOPPLER_FILTER)
305                 {
306                     dopplerCurve.AddPoint(readingTime,
                        tag.RfDopplerFrequency);
307                 }
308                 else
309                 {
310                     //dopplerCurve.AddPoint(readingTime,
                        dopplerCurve.GetCurveLastValue());
311                 }
312
313             }
314             catch (Exception e)
315             {
316                 // Handle .NET errors.
317                 Console.WriteLine("Exception : {0}", e.Message);
318             }
319         }
320
321         /// <summary>
322         /// Setter Ambiente
323         /// </summary>
324         public void SetCurrAmbient(Ambient currAmb)
325         {
326             if (currAmb == null)
327             {
328                 return;
329             }
330             currentAmbient = currAmb;
331         }
332
333         /// <summary>
334         /// Getter Ambiente
335         /// </summary>
336         public Ambient GetCurAmbient()
337         {
338             return currentAmbient;
339         }
340
341         /// <summary>
342         /// Setter Ambiente
343         /// TODO
344         /// </summary>
345         public void SetAmbient(Ambient ambientGuess, int i)
346         {
347             switch (i)
348             {
349                 case 0:
350                     ambPeakTimes = ambientGuess;
351                     break;
352                 case 1:
353                     ambPeakTimeAndMag = ambientGuess;
354                     break;
355                 case 2:
356                     ambLastRSSI = ambientGuess;
```

```
357                     break;
358                 case 3:
359                     ambRSSIMean = ambientGuess;
360                     break;
361                 case 4:
362                     ambRSSIMedian = ambientGuess;
363                     break;
364                 case 5:
365                     ambDopplerEffect = ambientGuess;
366                     break;
367                 case 6:
368                     ambDopplerAndRSSI = ambientGuess;
369                     break;
370             }
371         }
372 
373         /// <summary>
374         /// Getter Ambiente
375         /// TODO
376         /// </summary>
377         public Ambient GetAmbient(int i)
378         {
379             switch (i)
380             {
381                 case 0:
382                     return ambPeakTimes;
383                 case 1:
384                     return ambPeakTimeAndMag;
385                 case 2:
386                     return ambLastRSSI;
387                 case 3:
388                     return ambRSSIMean;
389                 case 4:
390                     return ambRSSIMedian;
391                 case 5:
392                     return ambDopplerEffect;
393                 case 6:
394                     return ambDopplerAndRSSI;
395                 default:
396                     return currentAmbient;
397             }
398         }
399 
400 
401 
402     }
403 }
404 
```