

https://dspace.org/

An investigation into visualisation and forecasting of real-time electrical consumption based on smart grid data

Mangisi, Rodwell

2015-05-05
http://hdl.handle.net/10539/17613

Downloaded from a DSpace repository. https://dspace.org/

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;

 5  namespace Models
 6  {
 7    public class PowerReading
 8    {
 9      public int id {
10      }

11      public float powerphase1 {
12      }

13      public float powerphase2 {
14      }

15      public float powerphase3 {
16      }

17      public float powertotal {
18      }

19      public DateTime metertimestamp {
20      }

21      public DateTime servertimestamp {
22      }

23      public int meter_id {
24      }

25      public Meter Meter {
26      }

27      public string Month
28      {
29          get { return servertimestamp.ToString("MMM"); }
30      }

31      public string Day
32      {
33          get { return servertimestamp.ToString("ddd"); }
34      }
35    }
36  }
```

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;
 5  using System.Data.Entity;
 6  using Models;

 7  namespace DataAccess
 8  {
 9    public class EnergyReadingRepository : DbContext
10    {
11      public Dictionary<int, Meter> MeterDetails = null;

12      public DbSet<DbEnergyReading> Readings {
13      }

14      public EnergyReadingRepository(Npgsql.NpgsqlConnection conn)
15      : base(conn, true)
16      {
17      }


18      protected override void OnModelCreating(DbModelBuilder modelBuilder)
19      {
20        modelBuilder.Entity<DbEnergyReading>().ToTable("energy_measurements", "
            public");
21      }


22      /// <summary>
23      /// Load EnergyReading from the past x minutes
24      /// </summary>
25      /// <param name="elapsedTime"></param>
26      /// <returns></returns>
27      public List<DbEnergyReading> LoadReadings(int elapsedTime, int meterID=
28    }
29  }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Xml;
6  using System.IO;
7  using System.Xml.Serialization;

8  namespace Configuration
9  {
10   public class Settings
11   {
12     public Settings()
13     {
14     }
```

```
15     public void CreateSettingsFile(ConfigSettings cs)
16     {
17       XmlWriterSettings settings = new XmlWriterSettings();
18       settings.Indent = true;

19       using (XmlWriter writer = XmlWriter.Create("settings.config", settings))
20       {

21         System.Xml.Serialization.XmlSerializer x = new System.Xml.Serialization
           XmlSerializer(cs.GetType());
22         x.Serialize(writer, cs);
23       }
24     }
```

```
25     /// <summary>
26     /// method that checks if file exists
27     /// </summary>
28     /// <returns></returns>
29     public bool DoesFileExist()
30     {
31       string file = "settings.config";
32       if (File.Exists(file))
33         return true;
34       else
35         return false;
36     }
```

```
37     public ConfigSettings ReadSettings()
38     {
39       XmlTextReader reader = new XmlTextReader("settings.config");
40       TextReader reader2 = new StreamReader("settings.config");

41       ConfigSettings cs = new ConfigSettings();
42       ConfigSettings cs2 = new ConfigSettings();
43       XmlSerializer serializer = new XmlSerializer(typeof(ConfigSettings));
44       cs = (ConfigSettings)serializer.Deserialize(reader);
45       cs2 = (ConfigSettings)serializer.Deserialize(reader2);
```

```
46          return cs;
47        }
48      }
49  }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;

5  namespace Configuration
6  {
7    public class ConfigSettings
8    {
9      public string ServerName {
10     }

11     public int PortNumber {
12     }

13     public string DatabaseName {
14     }

15     public string DatabaseUserName {
16     }

17     public string Password {
18     }
19   }
20 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;

5  namespace Models.Enumerations
6  {
7    /// <summary>
8    /// The campus open status.
9    /// </summary>
10   public enum eCampusStatus
11   {
12     CampusClosed = 0,
13     CampusOpen = 1,
14     Special = 2,
15   }
16 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.ComponentModel.DataAnnotations.Schema;

6  namespace DataAccess
7  {
8    [Table("locations", Schema = "public")]
9    public class DbLocation
10   {
11     public int id {
12     }
13     public string name {
14     }
15     public DbLocation()
16     {
17     }
18   }
19 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;

5  namespace Models
6  {
7    public class User
8    {
9      public string UserName {
10     }

11     public string Password {
12     }

13     public User(string name, string password)
14     {
15       this.UserName = name;
16       this.Password = password;
17     }

18   }
19 }
```

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;
 5  using System.Data.Entity;
 6  using Models;
 7  using System.Data.Common;
 8  using Npgsql;

 9  namespace DataAccess
10  {
11    public class MeterRepository : DbContext
12    {
13      // private List<Location> allLocations=null;
14      private Dictionary<int, Location> allLocations = null;
15      private Dictionary<int, MeterStatus> allMeterStatuses = null;
16      public Dictionary<int, Meter> allMeterDetails = null;

17      public DbSet<DbMeter> Meters {
18      }

19      public DbSet<DbLocation> Locations {
20      }

21      public DbSet<DbMeterStatus> MeterStatuses {
22      }

23      public MeterRepository(DbConnection conn)
24      : base(conn,true )
25      {
26      }


27      protected override void OnModelCreating(DbModelBuilder modelBuilder)
28      {
29        //  Database.SetInitializer<MeterRepository>(null);
30        // Map to the correct  Database tables
31        modelBuilder.Entity<DbMeter>().ToTable("meters", "public");
32        modelBuilder.Entity<DbLocation>().ToTable("locations", "public");
33        modelBuilder.Entity<DbMeterStatus>().ToTable("meter_statuses", "public");
34      }


35      public List<Meter> LoadMeters()
36      {
37        List<Meter> meters = new List<Meter>();
38        // this.LoadLocations(); methdo depricated as the locations table is no
    longer in use
39        this.LoadMeterStatuses();

40        allMeterDetails = new Dictionary<int, Meter>();
41        var selectedMeters = from meter in this.Meters select meter;
42        int count = selectedMeters.Count();
43        foreach (var item in selectedMeters)
44        {
45          Meter m = new Meter();
```

```
46          m.id = item.id;
47          m.Location = item.name;//allLocations[item.location_id];
48          m.MeterStatus = allMeterStatuses[item.meter_status_id];
49          m.serialno = item.serialno;
50          meters.Add(m);
51          allMeterDetails.Add(m.id, m);
52        }
53        return meters;
54      }

55      public void LoadLocations()
56      {
57        allLocations = new Dictionary<int, Location>();
58        var selectedLocations = from location in this.Locations select location;
59        int count = selectedLocations.Count();
60        foreach (var item in selectedLocations)
61        {
62          Location l = new Location();
63          l.id = item.id;
64          l.name = item.name;
65          allLocations.Add(l.id, l);

66        }
67      }

68      public void LoadMeterStatuses()
69      {
70        allMeterStatuses = new Dictionary<int, MeterStatus>();
71        var selectedStatuses = from status in this.MeterStatuses select status;
72        int count = selectedStatuses.Count();
73        foreach (var item in selectedStatuses)
74        {
75          MeterStatus ms = new MeterStatus();
76          ms.id = item.id;
77          ms.name = item.name;
78          allMeterStatuses.Add(ms.id, ms);
79        }
80        selectedStatuses = null;
81      }

82    }
83  }
```

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;
 5  using Models.Enumerations;

 6  namespace Models
 7  {
 8    /// <summary>
 9    /// Class to model the baseline variable
10    /// </summary>
11    public class BaselineVariable
12    {
13      public int HalfHourCycle {
14      }
15      public eCampusStatus CampusStatus  {
16      }
17      public DayOfWeek WeekDay {
18      }
19      public int Month {
20      }
21      public float BaselineValue {
22      }
23    }
24  }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;

5  namespace DataAccess
6  {
7    public class DbEnergyReading
8    {
9      public int id {
10     }

11     public int meter_id {
12     }

13     public bool is_estimate {
14     }

15     public float value {
16     }

17     public DateTime measurement_timestamp {
18     }

19     public string measurement_unit {
20     }
21   }
22 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Windows.Forms;
9  using System.Xml;
10 using Configuration;
11 using System.Data.Entity;
12 using DataAccess;
13 using Models;
14 using System.IO;
15 using Npgsql;
16 using System.Data.Common;

17 namespace EnergyManagementDashboard
18 {
19   public partial class FrmDBSettings : Form
20   {
21     private MeterRepository meterRepository;
22     private NpgsqlConnection conn;

23     public FrmDBSettings()
24     {
25       InitializeComponent();
26     }
```

```
27     private void simpleButton1_Click(object sender, EventArgs e)
28     {
29       NpgsqlConnectionStringBuilder csb = new NpgsqlConnectionStringBuilder();
30       csb.Database = textEditDbName.Text;
31       csb.Host = textEditServerName.Text;
32       csb.UserName = textEditDbUserName.Text; // must match Common Name of
   client certificate
33       csb.Password = textEditPassword.Text;
34       csb.SSL = true;
35       csb.Port = Convert.ToInt32(spinEditPortNumber.EditValue);
36       csb.SslMode = SslMode.Require;

37       conn = new NpgsqlConnection(csb.ConnectionString);

38       Settings settings = new Settings();
39       ConfigSettings cS = new ConfigSettings();
40       cS.ServerName = textEditServerName.Text;
41       cS.PortNumber = Convert.ToInt32(spinEditPortNumber.EditValue);
42       cS.DatabaseName = textEditDbName.Text;
43       cS.DatabaseUserName = textEditDbUserName.Text;
44       cS.Password = textEditPassword.Text;

45       SetUpRepositorys(conn);
46       try
47       {
48         List<Meter> Meters = meterRepository.LoadMeters();
```

```
49          settings.CreateSettingsFile(cS);
50          MessageBox.Show("Database connection successful!");
51          conn.Close();
52          Close();
53        }
54        catch (Npgsql.NpgsqlException ex)
55        {
56          MessageBox.Show("Invalid database settings.Please check your settings!/
              Detailed error:" + ex.ToString(),
57          "Wits Dashboard", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
58        }
59      }
```

```
60      private void SetUpRepositorys(NpgsqlConnection conn)
61      {
62        Database.SetInitializer<MeterRepository>(null);
63        meterRepository = new MeterRepository(conn);
64      }
```

```
65      public void updateConfigFile(string con)
66      {
67        //updating config file
68        XmlDocument XmlDoc = new XmlDocument();
69        //Loading the Config file
70        XmlDoc.Load(AppDomain.CurrentDomain.SetupInformation.ConfigurationFile);
71        foreach (XmlElement xElement in XmlDoc.DocumentElement)
72        {
73          if (xElement.Name == "connectionStrings")
74          {
75            //setting the coonection string
76            xElement.FirstChild.Attributes[1].Value = con;
77            xElement.FirstChild.NextSibling.Attributes[1].Value=con;
78            xElement.FirstChild.NextSibling.NextSibling.Attributes[1].Value = con;
79            xElement.FirstChild.NextSibling.NextSibling.NextSibling.Attributes[1]
                Value = con;

80          }
81        }
82        //writing the connection string in config file
83        XmlDoc.Save(AppDomain.CurrentDomain.SetupInformation.ConfigurationFile);
84        XmlDoc.Save("App.config");

85        if (File.Exists("App.config"))
86          MessageBox.Show("yeah");
87        else
88        {
89          MessageBox.Show("no!!");
90        }
91      }
```

```
92      private void FrmDBSettings_Load(object sender, EventArgs e)
93      {
94      }
```

```
95     }
96   }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;

5  namespace DataAccess
6  {
7    public class DbPowerReading
8    {
9      public int id {
10     }

11     public float powerphase1 {
12     }

13     public float powerphase2 {
14     }

15     public float powerphase3 {
16     }

17     public float powertotal {
18     }

19     public DateTime? metertimestamp {
20     }

21     public DateTime? servertimestamp {
22     }

23     public int meter_id {
24     }
25   }
26 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Text;
7  using System.Windows.Forms;
8  using DevExpress.XtraBars;
9  using DataAccess;
10 using System.Data.Entity;
11 using DevExpress.XtraCharts;
12 using GMap.NET;
13 using System.Linq;
14 using Models;
15 using DevExpress.XtraBars.Docking;
16 using DevExpress.XtraPivotGrid;
17 using Models.Enumerations;
18 using DevExpress.XtraLayout.Utils;
19 using DevExpress.XtraEditors;
20 using GMap.NET.WindowsForms;
21 using EnergyManagementDashboard.UIHelpers;
22 using Dashboard_Control;
23 using Configuration;
24 using Npgsql;
25 using System.Net;
26 using System.IO;
27 using System.Xml;

28 namespace EnergyManagementDashboard
29 {
30   /// <summary>
31   /// class to model the main form for the app
32   /// </summary>
33   public partial class FrmMain : DevExpress.XtraBars.Ribbon.RibbonForm
34   {
35     /// <summary>
36     /// file reader
37     /// </summary>
38     private BaselineFileReader FileReader;
39     /// <summary>
40     /// instance of calculator control
41     /// </summary>
42     private AlgorithmCalculations AlgorithmCalculator;
43     private MeterRepository meterRepository;
44     private PowerReadingRepository readingRepository;
45     private EnergyReadingRepository energyReadingRepository;
46     private EnergyWindowsRepository energyWindowsRepository;
47     private List<Meter> Meters;
48     private List<PowerReading> Readings;
49     private List<DbEnergyReading> EnergyReadings;
50     private bool isFirstTimeSingleMeter = true;
51     private bool isFirstTimeMultiMeter = true;
52     private int initialTimeInterval = 600;
53     private int initialEnergyTimeInterval = 600;
54     private int meterID = 1;
55     private int shortTimeInterval = 5;
```

```
56      public bool hasForecastBeenGivenForThisHour = false;
57      private List<ChartControl> SplitScreenChartControls = new List<ChartControl
          ();
58      private bool isPreviousPeackAvailable = false;
59      private NpgsqlConnection conn;

60      public float PreviousPeak {
61      }

62      public FrmMain()
63      {
64        InitializeComponent();
65      }


66      private void FrmMain_Load(object sender, EventArgs e)
67      {
68        DevExpress.LookAndFeel.UserLookAndFeel.Default.SkinName = "Caramel";
69      }


70      /// <summary>
71      /// Method that sets up all main form data
72      /// </summary>
73      public void LoadMainForm(ConfigSettings cS)
74      {
75        FileReader = new BaselineFileReader();

76        List<BaselineVariable> list = FileReader.GetVariables();
77        AlgorithmCalculator = new AlgorithmCalculations(list);
78        CreateConnection(cS);
79        SetUpRepositorys();
80        SetUpMultiMeterArchiveUI();
81        dockPanelSingleMeterLivePoower.Dock = DockingStyle.Fill;
82        SetUpChartTypes();
83        try
84        {
85          LoadMeters();
86        }
87        catch (Exception ex)
88        {
89          MessageBox.Show("No Available Database connection:\n" + ex.ToString());
90        }
91      }


92      private void CreateConnection(ConfigSettings cS)
93      {
94        NpgsqlConnectionStringBuilder csb = new NpgsqlConnectionStringBuilder();
95        csb.Database = cS.DatabaseName;
96        csb.Host = cS.ServerName;
97        csb.UserName = cS.DatabaseUserName; // must match Common Name of client
    certificate
98        csb.Password =cS.Password;
99        csb.SSL = true;
100       csb.Port = cS.PortNumber;
```

```
101         csb.SslMode = SslMode.Require;

102         conn = new NpgsqlConnection(csb.ConnectionString);
103       }

104       private void SetUpRepositorys()
105       {
106         Database.SetInitializer<MeterRepository>(null);
107         Database.SetInitializer<PowerReadingRepository>(null);
108         Database.SetInitializer<EnergyReadingRepository>(null);
109         Database.SetInitializer<EnergyWindowsRepository>(null);

110         meterRepository = new MeterRepository(conn);
111         readingRepository = new PowerReadingRepository(conn);
112         energyReadingRepository = new EnergyReadingRepository(conn);
113         energyWindowsRepository = new EnergyWindowsRepository(conn);
114         Meters = new List<Meter>();
115         Readings = new List<PowerReading>();
116         EnergyReadings = new List<DbEnergyReading>();
117         // repositoryMeterItemLookUpEdit.DataSource = Meters;
118         repositoryItemGridLookUpEditAllMeters.DataSource = Meters;
119         // gridControlPowerReadings.DataSource = Readings;

120       }

121       private void LoadMeters()
122       {
123         //try
124         //    {

125         Meters = meterRepository.LoadMeters();
126         repositoryMeterItemLookUpEdit.DataSource = Meters;
127         repositoryItemGridLookUpEditAllMeters.DataSource = Meters;
128         energyWindowsRepository.meters = meterRepository.allMeterDetails;
129         // }
130         //   catch (Npgsql.NpgsqlException e)
131         // {

132         //   MessageBox.Show("No Database connectivity"+"-"+e.Message.ToString());
133         //  }

134       }

135       private void LoadReadings()
136       {
137         readingRepository.LoadReadings(4, 1);
138         Readings = readingRepository.LoadMonthlyReadings(90, 0, 0, 1);
139         gridControlPowerReadings.DataSource = Readings;
140       }

141       /// <summary>
142       /// Method that loads the map provider
```

```
143       /// </summary>
144       private void LoadGoogleMap()
145       {
146         gmap.MapProvider = GMap.NET.MapProviders.GoogleMapProvider.Instance;
147         GMap.NET.GMaps.Instance.Mode = GMap.NET.AccessMode.ServerOnly;
148         gmap.Position = new PointLatLng(-26.191693, 28.026967);
149         gmap.Zoom = 18;
150       }
```

```
151       private PointLatLng Location(int locationID)
152       {
153         PointLatLng point = new PointLatLng(-26.191693, 28.026967);
154         String path = "";
155         switch (locationID)
156         {
157           case 1:
158             point = new PointLatLng(-26.19181, 28.026123);
159             path = "raikesRoad.jpg";
160             AddCustomMarkerToMap(point, path);
161             break;
162           case 2:
163             point = new PointLatLng(-26.191719, 28.031968);
164             path = "GateHouse.jpg";
165             AddCustomMarkerToMap(point, path);
166             break;
167           case 3:
168             point = new PointLatLng(-26.177383, 28.046684);
169             path = "medicalSchool.jpg";
170             AddCustomMarkerToMap(point, path);
171             break;
172           case 4:
173             point = new PointLatLng(-26.177383, 28.046684);
174             path = "medicalSchool.jpg";
175             AddCustomMarkerToMap(point, path);
176             break;
177           case 5:
178             point = new PointLatLng(-26.177383, 28.046684);
179             path = "medicalSchool.jpg";
180             AddCustomMarkerToMap(point, path);
181             break;
182           case 6:
183             point = new PointLatLng(-26.178399, 28.04206);
184             path = "WECEducationCampus.jpg";
185             AddCustomMarkerToMap(point, path);
186             break;
187           case 7:
188             point = new PointLatLng(-26.179901, 28.037543);
189             path = "EOHresPKV.jpg";
190             AddCustomMarkerToMap(point, path);
191             break;
192           case 8:
193             point = new PointLatLng(-26.191825, 28.031985);
194             path = "commerceLibrary.jpg";
195             AddCustomMarkerToMap(point, path);
196             break;
```

```
197         case 9://Gemni lab...put correct icon
198             point = new PointLatLng(-26.191719, 28.031968);
199             path = "commerceLibrary.jpg";
200             AddCustomMarkerToMap(point, path);
201             break;
202         default:
203             point = new PointLatLng(-26.189408, 28.026053);
204             path = "raikesRoad.jpg";
205             AddCustomMarkerToMap(point, path);
206             break;
207         }
208         return point;
209       }
```

```
210       private void AddCustomMarkerToMap(PointLatLng point, string path)
211       {
212         gmap.Overlays.Clear();
213         GMapOverlay top = new GMapOverlay(gmap, "top");
214         gmap.Overlays.Add(top);
215         Image marker = Image.FromFile(path);
216         GMapCustomMarker gCm = new GMapCustomMarker(point, marker);
217         top.Markers.Add(gCm);
218       }
```

```
219       private void btnLauchGISMap_ItemClick(object sender, ItemClickEventArgs e)
220       {
221         if (!dockPanelMap.IsMdiDocument)
222         {
223           LoadGoogleMap();
224           dockPanelMap.DockAsMdiDocument();
225           // AddCustomMarkerToMap();
226         }
227       }
```

```
228       private void AddCustomMarkerToMap()
229       {
230         GMapOverlay top = new GMapOverlay(gmap, "top");
231         gmap.Overlays.Add(top);
232         //gmap.Overlays[0].Markers.Add(
233         PointLatLng position = new PointLatLng(-26.191693, 28.026967);
234         Image test = Image.FromFile("Dependencies\\launch6.jpg");
235         GMapCustomMarker ddasd = new GMapCustomMarker(position, test);
236         top.Markers.Add(ddasd);
237       }
```

```
238       /// <summary>
239       /// Method to load all single meter readings
240       /// </summary>
241       /// <param name="initialTimeInterval"></param>
242       /// <param name="initialEnergyTimeInterval"></param>
243       /// <param name="meterID"></param>
244       /// <param name="shortTimeInterval"></param>
```

```
245     private void LoadAllSingleMeterReadings(int initialTimeInterval, int
          initialEnergyTimeInterval, int meterID, int shortTimeInterval)
246     {
247       Series series = chartControlPowerReadings.Series[0];
248       if (series == null)
249         return;
250       Series seriesEnergy = chartControlEnergyReadings.Series[0];

251       if (seriesEnergy == null)
252         return;
253       List<DbPowerReading> currentPowerReadings = null;
254       List<DbEnergyReading> currentEnergyReadings = null;
255       if (isFirstTimeSingleMeter)
256       {
257         currentPowerReadings = RefreshSingleMeterPowerReadings(
            initialTimeInterval, meterID, currentPowerReadings);
258         currentEnergyReadings = RefreshSingleMeterEnergyReadings(
            initialEnergyTimeInterval, meterID, currentEnergyReadings);
259         GetPrevioiusPeakForMeter(meterID);
260       }
261       else
262       {

263         currentPowerReadings = RefreshSingleMeterPowerReadings(shortTimeInterval
            , meterID, currentPowerReadings);
264         currentEnergyReadings = RefreshSingleMeterEnergyReadings(
            shortTimeInterval, meterID, currentEnergyReadings);
265         WarnPeakEnergyApproaching(currentEnergyReadings);
266       }
267       if (currentPowerReadings != null)
268       {
269         SeriesPoint[] pointsToUpdate = new SeriesPoint[currentPowerReadings.
            Count];
270         for (int i = 0; i < currentPowerReadings.Count; i++)
271         {
272           DbPowerReading r = currentPowerReadings[i];
273           pointsToUpdate[i] = new SeriesPoint(r.metertimestamp, r.powertotal);

274         }
275         series.Points.AddRange(pointsToUpdate);
276       }
277       if (currentEnergyReadings != null)
278       {
279         SeriesPoint[] pointsToUpdate = new SeriesPoint[currentEnergyReadings.
            Count];
280         for (int i = 0; i < currentEnergyReadings.Count; i++)
281         {
282           DbEnergyReading r = currentEnergyReadings[i];
283           pointsToUpdate[i] = new SeriesPoint(r.measurement_timestamp, r.value);

284         }
285         seriesEnergy.Points.AddRange(pointsToUpdate);
286       }
287     }
```

```
288     /// <summary>
289     /// Method to check if any of the values are approachig peak and give a
          warning.
290     /// </summary>
291     /// <param name="currentEnergyReadings"></param>
292     private void WarnPeakEnergyApproaching(List<DbEnergyReading>
          currentEnergyReadings)
293     {
294       int percentage = Convert.ToInt32(spnEditPercentageToPeak.EditValue);
295       double fraction = (double)percentage / (100);
296       int peakValue = Convert.ToInt32(spnEditPeakEnergyLine.EditValue);
297       double warn = peakValue - (fraction * peakValue);
298       int percentage1 = Convert.ToInt32(spnEditPercentageToPeak.EditValue);
299       if (currentEnergyReadings.Any(x => x.value >= warn))
300       {
301         MessageBox.Show("The energy value is within \"" + percentage1.ToString(
              + "%\"of set limit value!", "Wits Dasboard",
302         MessageBoxButtons.OK, MessageBoxIcon.Warning);
303       }

304         DateTime currentTime = DateTime.Now;

305         WebRequest req = HttpWebRequest.Create("
            http://api.openweathermap.org/data/2.5/weather?q=Johannesburg&mode=xml&units=metri
            ");
306         WebResponse res = req.GetResponse();
307         StreamReader sr = new StreamReader(res.GetResponseStream());
308         string str = sr.ReadToEnd();
309         XmlDocument doc = new XmlDocument();
310         doc.LoadXml(str);
311         XmlNodeList parentNode = doc.GetElementsByTagName("current");
312         string temperetureValue ="";
313         foreach (XmlNode item in parentNode)
314         {
315           temperetureValue = item.SelectSingleNode("temperature").Attributes.
              GetNamedItem("value").InnerText;// ("value").InnerText;
316         }
317         float tempValue=0;
318         if (float.TryParse(temperetureValue, out tempValue) &&
            isPreviousPeackAvailable)
319         {

320           float forecastLoad = AlgorithmCalculator.getBaselineValue((currentTime.
              Hour) * 2, eCampusStatus.Special, currentTime.DayOfWeek, currentTime.
              Month);
321           double forecast = AlgorithmCalculator.CalculateForecastLoadForCycle(
              tempValue, PreviousPeak, forecastLoad);
322           MessageBox.Show("The forecast peak load in this cycle is  \"" + forecas
              .ToString() + "\"kWh", "Wits Dasboard",
323           MessageBoxButtons.OK, MessageBoxIcon.Warning);
324         }
325     }


326     /// <summary>
327     /// Method to setu up split screen chart controls
```

```
328       /// </summary>
329       private void SetUpSplitScreenChartControls()
330       {
331         SplitScreenChartControls.Add(chartControlSplit1);
332         SplitScreenChartControls.Add(chartControlSplit2);
333         SplitScreenChartControls.Add(chartControlSplit3);
334         SplitScreenChartControls.Add(chartControlSplit4);
335       }

336       private void LoadAllMultiMeterReadings(int initialTimeInterval, int
            initialEnergyTimeInterval, int shortTimeInterval)
337       {
338         Series series = chartControlPowerReadings.Series[0];
339         if (series == null)
340           return;
341         Series seriesEnergy1 = chartControlSplit1.Series[0];
342         Series seriesEnergy2 = chartControlSplit2.Series[0];
343         Series seriesEnergy3 = chartControlSplit3.Series[0];
344         Series seriesEnergy4 = chartControlSplit4.Series[0];
345         Series seriesEnergy5 = chartControlSplit5.Series[0];
346         Series seriesEnergy6 = chartControlSplit6.Series[0];
347         Series seriesEnergy7 = chartControlSplit7.Series[0];
348         Series seriesEnergy8 = chartControlSplit8.Series[0];
349         List<DbPowerReading> currentPowerReadings = null;
350         List<DbEnergyReading> currentEnergyReadings = null;
351         if (isFirstTimeMultiMeter)
352         {
353           currentPowerReadings = RefreshMultimeterPowerReadings(
                initialTimeInterval, currentPowerReadings);
354           currentEnergyReadings = RefreshMultiMeterEnergyReadings(
                initialEnergyTimeInterval, currentEnergyReadings);

355         }
356         else
357         {

358           currentPowerReadings = RefreshMultimeterPowerReadings(shortTimeInterval
                currentPowerReadings);//
    readingRepository.LoadReadings(shortTimeInterval, meterID);
359           currentEnergyReadings = RefreshMultiMeterEnergyReadings(
                shortTimeInterval, currentEnergyReadings);//
    energyReadingRepository.LoadReadings(shortTimeInterval, meterID);
360         }

361         //ToDo rodwell... Get distinct meter IDs...create energy reading lists for
              each meter. Populate a chart control for each list
362         //hide unpopulated charts.If list of Ids is greater than the initial 4
    then create a chart control on the fly.add meter ID to the chartcontrol
363         //tag so that when u refresh u know which chart a reading belongs to.


364         List<DbEnergyReading> one = new List<DbEnergyReading>();//put
    instantiation in the first time bool...so that u dont delete vales on
    refress..or is it important since we dont clear actuall series?
365         List<DbEnergyReading> two = new List<DbEnergyReading>();
```

```
366         List<DbEnergyReading> three = new List<DbEnergyReading>();
367         List<DbEnergyReading> four = new List<DbEnergyReading>();
368         List<DbEnergyReading> five = new List<DbEnergyReading>();
369         List<DbEnergyReading> six = new List<DbEnergyReading>();
370         List<DbEnergyReading> seven = new List<DbEnergyReading>();
371         List<DbEnergyReading> eight = new List<DbEnergyReading>();

372         foreach (DbEnergyReading item in currentEnergyReadings)
373         {
374             switch (item.meter_id)
375             {
376                 case 1:
377                     one.Add(item);
378                     break;
379                 case 2:
380                     two.Add(item);
381                     break;
382                 case 3:
383                     three.Add(item);
384                     break;
385                 case 4:
386                     four.Add(item);
387                     break;
388                 case 5:
389                     five.Add(item);
390                     break;
391                 case 6:
392                     six.Add(item);
393                     break;
394                 case 7:
395                     seven.Add(item);
396                     break;
397                 case 8:
398                     eight.Add(item);
399                     break;
400                 default:
401                     break;
402             }

403         }
404         //if (currentPowerReadings != null)
405         //{
406         //  SeriesPoint[] pointsToUpdate = new
    SeriesPoint[currentPowerReadings.Count];
407         //  for (int i = 0; i < currentPowerReadings.Count; i++)
408         //  {
409         //    DbPowerReading r = currentPowerReadings[i];
410         //    pointsToUpdate[i] = new SeriesPoint(r.metertimestamp, r.powertotal);

411         //  }
412         //  series.Points.AddRange(pointsToUpdate);
413         //}

414         if (isFirstTimeMultiMeter)
415         {
416             if (one.Count == 0)
```

```
417         {
418           layoutControlItemchartControlSplit1.Visibility = LayoutVisibility.
                Never;
419           splitterItem1And8.Visibility = LayoutVisibility.Never;
420           Meters.FirstOrDefault(x => x.id == 1).IsShown = false;
421         }
422         if (two.Count == 0)
423         {
424           layoutControlItemchartControlSplit2.Visibility = LayoutVisibility.
                Never;
425           splitterItem7and2.Visibility = LayoutVisibility.Never;
426           Meters.FirstOrDefault(x => x.id == 2).IsShown = false;
427         }
428         if (three.Count == 0)
429         {
430           layoutControlItemchartControlSplit3.Visibility = LayoutVisibility.
                Never;
431           splitterItem4and3.Visibility = LayoutVisibility.Never;
432           Meters.FirstOrDefault(x => x.id == 3).IsShown = false;

433         }
434         if (four.Count == 0)
435         {
436           layoutControlItemchartControlSplit4.Visibility = LayoutVisibility.
                Never;
437           if (!layoutControlItemchartControlSplit3.Visible)
438           {
439             splitterItem4and3.Visibility = LayoutVisibility.Never;
440           }
441           if (!layoutControlItemchartControlSplit8.Visible)
442           {
443             splitterItem8and4.Visibility = LayoutVisibility.Never;
444           }

445           Meters.FirstOrDefault(x => x.id == 4).IsShown = false;
446         }
447         if (five.Count == 0)
448         {
449           layoutControlItemchartControlSplit5.Visibility = LayoutVisibility.
                Never;
450           if (!layoutControlItemchartControlSplit7.Visible)
451           {
452             splitterItem5and7.Visibility = LayoutVisibility.Never;
453           }
454           if (!layoutControlItemchartControlSplit6.Visible)
455           {
456             splitterItem6and5.Visibility = LayoutVisibility.Never;
457           }

458           Meters.FirstOrDefault(x => x.id == 5).IsShown = false;
459         }
460         if (six.Count == 0)
461         {
462           layoutControlItemchartControlSplit6.Visibility = LayoutVisibility.
                Never;
463           splitterItem6and5.Visibility = LayoutVisibility.Never;
```

```
464             Meters.FirstOrDefault(x => x.id == 6).IsShown = false;
465         }
466         if (seven.Count == 0)
467         {
468             layoutControlItemchartControlSplit7.Visibility = LayoutVisibility.
                  Never;
469             if (!layoutControlItemchartControlSplit5.Visible)
470             {
471                 splitterItem5and7.Visibility = LayoutVisibility.Never;
472             }
473             if (!layoutControlItemchartControlSplit2.Visible)
474             {
475                 splitterItem7and2.Visibility = LayoutVisibility.Never;
476             }

477             Meters.FirstOrDefault(x => x.id == 7).IsShown = false;
478         }
479         if (eight.Count == 0)
480         {
481             layoutControlItemchartControlSplit8.Visibility = LayoutVisibility.
                  Never;
482             if (!layoutControlItemchartControlSplit1.Visible)
483             {
484                 splitterItem1And8.Visibility = LayoutVisibility.Never;
485             }
486             if (!layoutControlItemchartControlSplit4.Visible)
487             {
488                 splitterItem8and4.Visibility = LayoutVisibility.Never;
489             }

490             //  Meters.FirstOrDefault(x => x.id == 8).IsShown = false;
491         }
492     }

493         AddMultiEnergyValuesToChatSeries(seriesEnergy1, one);
494         AddMultiEnergyValuesToChatSeries(seriesEnergy2, two);
495         AddMultiEnergyValuesToChatSeries(seriesEnergy3, three);
496         AddMultiEnergyValuesToChatSeries(seriesEnergy4, four);
497         AddMultiEnergyValuesToChatSeries(seriesEnergy5, five);
498         AddMultiEnergyValuesToChatSeries(seriesEnergy6, six);
499         AddMultiEnergyValuesToChatSeries(seriesEnergy7, seven);
500         AddMultiEnergyValuesToChatSeries(seriesEnergy8, eight);
501     }
```

```
502     /// <summary>
503     /// method to add multi energy values to chart series
504     /// </summary>
505     /// <param name="seriesEnergy1"></param>
506     /// <param name="one"></param>
507     private static void AddMultiEnergyValuesToChatSeries(Series seriesEnergy1,
          List<DbEnergyReading> one)
508     {
509         if (one.Count > 0)
510         {
511             SeriesPoint[] pointsToUpdate = new SeriesPoint[one.Count];
```

```
512         for (int i = 0; i < one.Count; i++)
513         {
514           DbEnergyReading r = one[i];
515           pointsToUpdate[i] = new SeriesPoint(r.measurement_timestamp, r.value);
516         }
517         seriesEnergy1.Points.AddRange(pointsToUpdate);
518       }
519     }
```

```
520     private List<DbEnergyReading> RefreshSingleMeterEnergyReadings(int
          initialTimeInterval, int meterID, List<DbEnergyReading>
          currentEnergyReadings)
521     {
522       timer1SingleMeter.Stop();
523       currentEnergyReadings = energyReadingRepository.LoadReadings(
            initialTimeInterval, meterID);
524       timer1SingleMeter.Interval = 15000;
525       isFirstTimeSingleMeter = false;
526       timer1SingleMeter.Start();
527       //

528       GetPrevioiusPeakForMeter(meterID);
529       //

530       return currentEnergyReadings;
531     }
```

```
532     /// <summary>
533     /// Method to get previous peak based on meter
534     /// </summary>
535     /// <param name="meterID"></param>
536     private void GetPrevioiusPeakForMeter(int meterID)
537     {
538       DateTime dateTimeFromThisDat = DateTime.Now.Subtract(new TimeSpan(0, 31,
            ));
539       DateTime dateTimeToThisDat = DateTime.Now;
540       int hours = Convert.ToInt32(spnEdtHours.EditValue);
541       int minutes = Convert.ToInt32(spnEdtMinutes.EditValue);
542       List<EnergyWindow> windows = energyWindowsRepository.
            LoadEnergyWindowForSingleMeter(eSelectionFormat.HoursMinutes,
            dateTimeFromThisDat, dateTimeToThisDat, hours, minutes, meterID);

543       if (windows.Count > 0)
544       {
545         PreviousPeak = (float)(windows[0].energy_consumed);
546         isPreviousPeackAvailable = true;
547       }
548       else
549       {
550         isPreviousPeackAvailable = false;
551         MessageBox.Show("The current Cycle will not be forecast as previous peak
              was not calculated!", "Wits Dasboard",
552         MessageBoxButtons.OK, MessageBoxIcon.Warning);
```

```
553         }
554     }

555     private List<DbEnergyReading> RefreshMultiMeterEnergyReadings(int
          initialTimeInterval, List<DbEnergyReading> currentEnergyReadings)
556     {
557         timerMultiMeterScreens.Stop();
558         currentEnergyReadings = energyReadingRepository.LoadReadings(
              initialTimeInterval);
559         timerMultiMeterScreens.Interval = 15000;
560         timerMultiMeterScreens.Start();
561         return currentEnergyReadings;
562     }

563     private List<DbPowerReading> RefreshSingleMeterPowerReadings(int
          initialTimeInterval, int meterID, List<DbPowerReading> currentReadings)
564     {
565         timer1SingleMeter.Stop();
566         currentReadings = readingRepository.LoadReadings(initialTimeInterval,
              meterID);
567         timer1SingleMeter.Interval = 15000;
568         isFirstTimeSingleMeter = false;
569         timer1SingleMeter.Start();
570         return currentReadings;
571     }

572     private List<DbPowerReading> RefreshMultimeterPowerReadings(int
          initialTimeInterval, List<DbPowerReading> currentReadings)
573     {
574         timerMultiMeterScreens.Stop();
575         currentReadings = readingRepository.LoadReadings(initialTimeInterval, -1);
576         timerMultiMeterScreens.Interval = 15000;
577         timerMultiMeterScreens.Start();
578         return currentReadings;
579     }

580     private void repositoryItemGridLookUpEditAllMetersItem_EditValueChanged(
          object sender, EventArgs e)
581     {
582         btnSingleMeterStartLiveReadings.Enabled = true;
583         btnSingleMeterStopLiveMetterReadings.Enabled = true;
584         if (dateTimeSingleTimeFromThisDate.EditValue != null &&
              dateTimeSingleMeterToThisDate.EditValue != null && (
              repositoryItemGridLookUpEditAllMetersItem.EditValue != null))
585         {
586             btnLaunchSingleMeterArchive.Enabled = true;
587             btnLaunchSingleMeterEnergRawEnergyReadings.Enabled = true;
588         }
589         if ((repositoryItemGridLookUpEditAllMetersItem.EditValue != null))
590         {
591             txtEditActiveMeter.EditValue = ((Meter)
                  repositoryItemGridLookUpEditAllMetersItem.EditValue).Location;//.name;
```

```
592         }
593     }

594     public void LoadMeterArchiveCharts()
595     {
596       // pivotGridControl1.DataSource = readingRepository.LoadMonthlyReadings(0,
            1, 1, 1);
597       if (meterRepository.allMeterDetails != null)
598       {
599         readingRepository.MeterDetails = meterRepository.allMeterDetails;
600       }
601       List<PowerReading> dings = readingRepository.LoadMonthlyReadings(2, 1, 1,
            1);
602       pivotGridControlSingleMeter.DataSource = dings;
603       chartControlMeterPowerArchive.DataSource = pivotGridControlSingleMeter;
604       //  Series seriesEnergy = chartControlMeterArchive.Series[0];
605       // SeriesPoint[] pointsToUpdate = new SeriesPoint[dings.Count];
606       /*for (int i = 0; i < dings.Count; i++)
607       {
608       PowerReading r = dings[i];
609       pointsToUpdate[i] = new SeriesPoint(r.metertimestamp, r.powertotal);

610     }-*/
611       // seriesEnergy.Points.AddRange(pointsToUpdate);

612     }

613     /// <summary>
614     /// Handler for loading single meter archive data
615     /// </summary>
616     /// <param name="sender"></param>
617     /// <param name="e"></param>
618     private void btnLaunchSingleMeterPowerArchive_ItemClick(object sender,
          ItemClickEventArgs e)
619     {
620       Cursor.Current = Cursors.WaitCursor;
621       if (meterRepository.allMeterDetails != null)
622       {
623         readingRepository.MeterDetails = meterRepository.allMeterDetails;


624         DateTime dateTimeFromThisDat = (DateTime)dateTimeSingleTimeFromThisDate.
            EditValue;
625         DateTime dateTimeToThisDat = (DateTime)dateTimeSingleMeterToThisDate.
            EditValue;
626         List<PowerReading> dings = readingRepository.LoadPreviouseReadings(
            dateTimeFromThisDat, dateTimeToThisDat, 1);
627         pivotGridControlSingleMeter.DataSource = dings;
628         chartControlMeterPowerArchive.DataSource = pivotGridControlSingleMeter;
629         gridControlPowerReadings.DataSource = dings;
630         dockPanelSingleMeterPowerArchive.DockAsMdiDocument();

631       }
632       Cursor.Current = Cursors.Default;
```

```
633     }

634     /// <summary>
635     /// handler for date changed event
636     /// </summary>
637     /// <param name="sender"></param>
638     /// <param name="e"></param>
639     private void dateTimeToThisDate_EditValueChanged(object sender, EventArgs e)
640     {
641         if (dateTimeSingleTimeFromThisDate.EditValue != null &&
              dateTimeSingleMeterToThisDate.EditValue != null && (
              repositoryItemGridLookUpEditAllMetersItem.EditValue != null))
642         {
643             btnLaunchSingleMeterArchive.Enabled = true;
644             btnLaunchSingleMeterEnergRawEnergyReadings.Enabled = true;
645         }
646     }

647     /// <summary>
648     /// Handler for launching all meter historical data
649     /// </summary>
650     /// <param name="sender"></param>
651     /// <param name="e"></param>
652     private void btnLaunchMultiMeterArchive_ItemClick(object sender,
          ItemClickEventArgs e)
653     {
654         if (meterRepository.allMeterDetails != null)
655         {

656             energyWindowsRepository.meters = meterRepository.allMeterDetails;


657             DateTime dateTimeFromThisDat = (DateTime)
                  dateTimeMultiMeterTimeFromThisDate.EditValue;
658             DateTime dateTimeToThisDat = (DateTime)dateTimeMultiMeterTimeToThisDate
                  EditValue;
659             int hours = Convert.ToInt32(spnEdtHours.EditValue);
660             int minutes = Convert.ToInt32(spnEdtMinutes.EditValue);
661             List<EnergyWindow> windows = energyWindowsRepository.
                  LoadEnergyWindowReadings(eSelectionFormat.DateTime,
                  dateTimeFromThisDat, dateTimeToThisDat, hours, minutes);
662             // pivotGridControlSingleMeter.DataSource = windows;
663             pivotGridControlAllMetersArchive.DataSource = windows;
664             // chartControlMeterArchive.DataSource = pivotGridControlSingleMeter;
665             chartControlAllMetersEnergyWindowArchive.DataSource =
                  pivotGridControlAllMetersArchive;
666             //  dockPanelAllMetersEnergyArchive.Show();
667             if (!dockPanelAllMetersEnergyWindowsArchive.IsMdiDocument)
668             {
669                 dockPanelAllMetersEnergyWindowsArchive.DockAsMdiDocument();
670             }
671         }
672     }
```

```
673     private void dateTimeMultiMeterTimeFromThisDate_EditValueChanged(object
          sender, EventArgs e)
674     {
675       if (dateTimeMultiMeterTimeFromThisDate.EditValue != null &&
            dateTimeMultiMeterTimeToThisDate != null)
676       {
677         btnLaunchMultiMeterEnergyWindowsArchive.Enabled = true;
678         btnLaunchMultiMeterEnergyReadingsArchive.Enabled = true;
679       }
680     }

681     private void dateTimeMultiMeterTimeToThisDate_EditValueChanged(object sende
          , EventArgs e)
682     {
683       if (dateTimeMultiMeterTimeFromThisDate.EditValue != null &&
            dateTimeMultiMeterTimeToThisDate != null)
684       {
685         btnLaunchMultiMeterEnergyWindowsArchive.Enabled = true;
686       }
687     }

688     /// <summary>
689     /// Method to set up multi meter archive UI
690     /// </summary>
691     private void SetUpMultiMeterArchiveUI()
692     {
693       chkEditUseDateSelection.EditValue = false;
694       chkEditUseTimeSelection.EditValue = true;
695       dateTimeMultiMeterTimeToThisDate.EditValue = DateTime.Today;
696       dateTimeMultiMeterTimeFromThisDate.EditValue = DateTime.Today;
697     }

698     private void chkEditUseDateSelection_EditValueChanged(object sender,
          EventArgs e)
699     {
700       ToggleSelectionOptionsForMultiMeter();
701     }

702     private void ToggleSelectionOptionsForMultiMeter()
703     {
704       this.chkEditUseDateSelection.EditValueChanged -= new System.EventHandler(
            this.chkEditUseDateSelection_EditValueChanged);
705       this.chkEditUseTimeSelection.EditValueChanged -= new System.EventHandler(
            this.chkEditUseTimeSelection_EditValueChanged);
706       chkEditUseDateSelection.EditValue = !(bool)chkEditUseTimeSelection.
            EditValue;
707       chkEditUseTimeSelection.EditValue = !(bool)chkEditUseDateSelection.
            EditValue;
708       this.chkEditUseDateSelection.EditValueChanged += new System.EventHandler(
            this.chkEditUseDateSelection_EditValueChanged);
709       this.chkEditUseTimeSelection.EditValueChanged += new System.EventHandler(
            this.chkEditUseTimeSelection_EditValueChanged);
```

```
710     }

711     private void chkEditUseTimeSelection_EditValueChanged(object sender,
          EventArgs e)
712     {
713       ToggleSelectionOptionsForMultiMeter();
714     }

715     /// <summary>
716     /// Handler to load the grid with energy readings.
717     /// </summary>
718     /// <param name="sender"></param>
719     /// <param name="e"></param>
720     private void btnLoadSingleMeterArchivedEnergyReadings_ItemClick(object
          sender, ItemClickEventArgs e)
721     {
722       Cursor.Current = Cursors.WaitCursor;
723       if (meterRepository.allMeterDetails != null)
724       {
725         //
726         pivotGridControlSingleMeterEnergyArchive.DataSource = null;
727         chartControlMeterEnergyArchive.DataSource = null;
728         gridControlRawEnergyReadings.DataSource = null;
729         pivotGridControlSingleMeterEnergyArchive.CollapseAll();
730         //
731         Meter selectedMeter = null;
732         energyReadingRepository.MeterDetails = meterRepository.allMeterDetails;
733         if (dateTimeSingleTimeFromThisDate.EditValue != null &&
              dateTimeSingleMeterToThisDate.EditValue != null && (
              repositoryItemGridLookUpEditAllMetersItem.EditValue != null))
734         {
735           selectedMeter = repositoryItemGridLookUpEditAllMetersItem.EditValue a
                Meter;
736         }
737         if (selectedMeter != null)
738         {
739           DateTime dateTimeFromThisDat = (DateTime)
                dateTimeSingleTimeFromThisDate.EditValue;
740           DateTime dateTimeToThisDat = (DateTime)dateTimeSingleMeterToThisDate.
                EditValue;
741           List<EnergyReading> dings = energyReadingRepository.
                LoadEnergyReadingsBasedOnDates(dateTimeFromThisDat,
                dateTimeToThisDat, (double)spnEditorEnergyPricePerKwhr.EditValue,
                selectedMeter.id);

742           if (dings.Count > 0)
743           {

744             pivotGridControlSingleMeterEnergyArchive.DataSource = dings;
745             chartControlMeterEnergyArchive.DataSource =
                  pivotGridControlSingleMeterEnergyArchive;
746             gridControlRawEnergyReadings.DataSource = dings;
747             documentManager.BeginUpdate();
748             if (!dockPanelRawEnergyReadings.IsMdiDocument)
```

```
749                 {
750                   dockPanelRawEnergyReadings.DockAsMdiDocument();
751                 }
752                 if (!dockPanelMeterEnergyReadingsArchive.IsMdiDocument)
753                 {
754                   dockPanelMeterEnergyReadingsArchive.DockAsMdiDocument();
755                 }
756                 documentManager.EndUpdate();
757               }
758               else
759               {
760                 MessageBox.Show("The selected meter does not have any readings in
                    the selected range!", "Wits Dasboard",
761                 MessageBoxButtons.OK, MessageBoxIcon.Error);
762               }
763             }
764           }

765           Cursor.Current = Cursors.Default;
766         }
```

```
767         /// <summary>
768         /// Handler to launch multi meter split screen Live readings.
769         /// </summary>
770         /// <param name="sender"></param>
771         /// <param name="e"></param>
772         private void btnMultiMeterLiveReadingsLaunch_ItemClick(object sender,
              ItemClickEventArgs e)
773         {
774           timerMultiMeterScreens.Enabled = false;
775           timerMultiMeterScreens.Stop();
776           SetUpRepositorys();
777           LoadMeters();
778           repositoryItemGridLookUpEditShowHideMeterCharts.DataSource = Meters;
779           //  isFirstTimeMultiMeter = true;
780           // Series series = chartControlPowerReadings.Series[0];
781           //Series energySeries = chartControlEnergyReadings.Series[0];
782           // series.Points.Clear();
783           //energySeries.Points.Clear();
784           //
785           chartControlSplit1.Series[0].Points.Clear();
786           chartControlSplit2.Series[0].Points.Clear();
787           chartControlSplit3.Series[0].Points.Clear();
788           chartControlSplit4.Series[0].Points.Clear();
789           chartControlSplit5.Series[0].Points.Clear();
790           chartControlSplit6.Series[0].Points.Clear();
791           chartControlSplit7.Series[0].Points.Clear();
792           chartControlSplit8.Series[0].Points.Clear();

793           //
794           initialTimeInterval = 60;
795           initialEnergyTimeInterval = 600;
796           //meterID = m.id;
797           shortTimeInterval = 5;
```

```
798         LoadAllMultiMeterReadings(initialTimeInterval, initialEnergyTimeInterval,
              shortTimeInterval);

799         timerMultiMeterScreens.Enabled = true;
800         timer1SingleMeter.Enabled = false;
801         timerMultiMeterScreens.Start();
802         isFirstTimeMultiMeter = false;

803         if (!dockPanelMultiMeterSplitLiveGraphs.IsMdiDocument)
804         {
805           dockPanelMultiMeterSplitLiveGraphs.DockAsMdiDocument();
806         }
807       }
```

```
808       private void timer1SingleMeter_Tick(object sender, EventArgs e)
809       {
810         timerMultiMeterScreens.Stop();//avoid both timers running same time
811         timer1SingleMeter.Stop();
812         Cursor.Current = Cursors.WaitCursor;
813         LoadAllSingleMeterReadings(initialTimeInterval, initialEnergyTimeInterval
              meterID, shortTimeInterval);
814         timer1SingleMeter.Start();
815         SwiftPlotDiagram diagram = (SwiftPlotDiagram)chartControlEnergyReadings.
              Diagram;
816         diagram.AxisY.ConstantLines[0].AxisValue = spnEditPeakEnergyLine.EditValu
              ;
817         SwiftPlotDiagram diagram1 = (SwiftPlotDiagram)chartControlPowerReadings.
              Diagram;
818         diagram1.AxisY.ConstantLines[0].AxisValue = spnEditPeakPowerValue.
              EditValue;
819         Cursor.Current = Cursors.Default;
820       }
```

```
821       private void timerMultiMeterScreens_Tick(object sender, EventArgs e)
822       {
823         timer1SingleMeter.Stop();//avoid two timers running at the same time
824         timerMultiMeterScreens.Stop();//lets not count loading time
825         Cursor.Current = Cursors.WaitCursor;
826         LoadAllMultiMeterReadings(initialTimeInterval, initialEnergyTimeInterval,
              shortTimeInterval);
827         Cursor.Current = Cursors.Default;
828       }
```

```
829       private void btnSingleMeterStopLiveMetterReadings_ItemClick(object sender,
            ItemClickEventArgs e)
830       {
831         //chartControlSplit1.Series[0].ChangeView(ViewType.SwiftPlot);

832         if (timer1SingleMeter.Enabled)
833         {
834           timer1SingleMeter.Stop();
835           btnSingleMeterStopLiveMetterReadings.Caption = "Resume Live Readings";
836         }
```

```
837         else
838         {
839           timer1SingleMeter.Enabled = true;
840           timer1SingleMeter.Start();
841           btnSingleMeterStopLiveMetterReadings.Caption = "Pause Live Readings";
842         }
843       }
```

```
844       private void btnMultiMeterLiveReadingsStop_ItemClick(object sender,
            ItemClickEventArgs e)
845       {
846         if (timerMultiMeterScreens.Enabled)
847         {
848           timerMultiMeterScreens.Enabled = false;
849           timerMultiMeterScreens.Stop();
850           btnMultiMeterLiveReadingsStop.Caption = "Resume Live Readings";
851         }
852         else
853         {
854           timerMultiMeterScreens.Enabled = true;
855           timerMultiMeterScreens.Start();
856           btnMultiMeterLiveReadingsStop.Caption = "Pause Live Readings";
857         }
858       }
```

```
859       private void gridLookUpEditShowHideMeterCharts_EditValueChanged(object
            sender, EventArgs e)
860       {
861         BarEditItem edit = sender as BarEditItem;
862         if (edit.EditValue.GetType() == typeof(Meter))
863         {
864           ToggleMeterChatDisplay(edit.EditValue as Meter);
865         }
866       }
```

```
867       private void ToggleMeterChatDisplay(Meter meter)
868       {
869         switch (meter.id)
870         {
871           case 1:

872             if (meter.IsShown)
873             {
874               layoutControlItemchartControlSplit1.Visibility = LayoutVisibility.
                    Never;

875               splitterItem1And8.Visibility = LayoutVisibility.Never;
876               meter.IsShown = false;
877             }
878             else
879             {
880               layoutControlItemchartControlSplit1.Visibility = LayoutVisibility.
                    Always;
```

```
881             if (layoutControlItemchartControlSplit8.Visible ||
                    layoutControlItemchartControlSplit4.Visible ||
                    layoutControlItemchartControlSplit3.Visible)
882             {
883               splitterItem1And8.Visibility = LayoutVisibility.Always;
884             }
885             meter.IsShown = true;
886           }

887           break;

888         case 2:
889           if (meter.IsShown)
890           {
891             layoutControlItemchartControlSplit2.Visibility = LayoutVisibility.
                  Never;
892             splitterItem7and2.Visibility = LayoutVisibility.Never;
893             meter.IsShown = false;
894           }
895           else
896           {
897             layoutControlItemchartControlSplit2.Visibility = LayoutVisibility.
                  Always;
898             if (layoutControlItemchartControlSplit7.Visible ||
                    layoutControlItemchartControlSplit5.Visible ||
                    layoutControlItemchartControlSplit6.Visible)
899             {
900               splitterItem7and2.Visibility = LayoutVisibility.Always;
901             }
902             if (layoutControlItemchartControlSplit6.Visible)
903             {
904               splitterItem6and5.Visibility = LayoutVisibility.Always;

905             }
906             meter.IsShown = true;
907           }
908           break;
909         case 3:
910           if (meter.IsShown)
911           {
912             layoutControlItemchartControlSplit3.Visibility = LayoutVisibility.
                  Never;
913             splitterItem4and3.Visibility = LayoutVisibility.Never;
914             meter.IsShown = false;

915           }
916           else
917           {
918             layoutControlItemchartControlSplit3.Visibility = LayoutVisibility.
                  Always;
919             if (layoutControlItemchartControlSplit8.Visible ||
                    layoutControlItemchartControlSplit4.Visible ||
                    layoutControlItemchartControlSplit1.Visible)
920             {
921               splitterItem4and3.Visibility = LayoutVisibility.Always;
922             }
```

```
923             if (layoutControlItemchartControlSplit1.Visible)
924             {
925               splitterItem1And8.Visibility = LayoutVisibility.Always;
926             }
927             meter.IsShown = true;
928           }
929           break;
930         case 4:
931           if (meter.IsShown)
932           {
933             layoutControlItemchartControlSplit4.Visibility = LayoutVisibility.
                  Never;
934             if (!layoutControlItemchartControlSplit3.Visible)
935             {
936               splitterItem4and3.Visibility = LayoutVisibility.Never;
937             }
938             if (!layoutControlItemchartControlSplit8.Visible)
939             {
940               splitterItem8and4.Visibility = LayoutVisibility.Never;
941             }
942             meter.IsShown = false;
943           }
944           else
945           {
946             layoutControlItemchartControlSplit4.Visibility = LayoutVisibility.
                  Always;
947             if (layoutControlItemchartControlSplit3.Visible)
948             {
949               splitterItem4and3.Visibility = LayoutVisibility.Always;
950             }
951             if (layoutControlItemchartControlSplit8.Visible)
952             {
953               splitterItem8and4.Visibility = LayoutVisibility.Always;
954             }
955             if (layoutControlItemchartControlSplit1.Visible)
956             {
957               splitterItem1And8.Visibility = LayoutVisibility.Always;
958             }
959             meter.IsShown = true;
960           }
961           break;
962         case 5:
963           if (meter.IsShown)
964           {
965             layoutControlItemchartControlSplit5.Visibility = LayoutVisibility.
                  Never;
966             if (!layoutControlItemchartControlSplit7.Visible)
967             {
968               splitterItem5and7.Visibility = LayoutVisibility.Never;
969             }
970             if (!layoutControlItemchartControlSplit6.Visible)
971             {
972               splitterItem6and5.Visibility = LayoutVisibility.Never;
973             }
```

```
974             meter.IsShown = false;
975           }
976           else
977           {
978             layoutControlItemchartControlSplit5.Visibility = LayoutVisibility.
                  Always;
979             if (layoutControlItemchartControlSplit7.Visible)
980             {
981               splitterItem5and7.Visibility = LayoutVisibility.Always;
982             }
983             if (layoutControlItemchartControlSplit6.Visible)
984             {
985               splitterItem6and5.Visibility = LayoutVisibility.Always;
986             }
987             meter.IsShown = true;
988           }
989           break;
990         case 6:
991           if (meter.IsShown)
992           {
993             layoutControlItemchartControlSplit6.Visibility = LayoutVisibility.
                  Never;
994             splitterItem6and5.Visibility = LayoutVisibility.Never;
995             meter.IsShown = false;
996           }
997           else
998           {
999             layoutControlItemchartControlSplit6.Visibility = LayoutVisibility.
                  Always;
1000            if (layoutControlItemchartControlSplit5.Visible ||
                  layoutControlItemchartControlSplit7.Visible ||
                  layoutControlItemchartControlSplit2.Visible)
1001            {
1002              splitterItem6and5.Visibility = LayoutVisibility.Always;
1003            }
1004            meter.IsShown = true;
1005          }
1006          break;
1007        case 7:
1008          if (meter.IsShown)
1009          {
1010            layoutControlItemchartControlSplit7.Visibility = LayoutVisibility.
                  Never;
1011            if (!layoutControlItemchartControlSplit5.Visible)
1012            {
1013              splitterItem5and7.Visibility = LayoutVisibility.Never;
1014            }
1015            if (!layoutControlItemchartControlSplit2.Visible)
1016            {
1017              splitterItem7and2.Visibility = LayoutVisibility.Never;
1018            }
1019            meter.IsShown = false;
1020          }
1021          else
1022          {
1023            layoutControlItemchartControlSplit7.Visibility = LayoutVisibility.
```

```
                Always;
1024            if (layoutControlItemchartControlSplit2.Visible)
1025            {
1026              splitterItem7and2.Visibility = LayoutVisibility.Always;
1027            }
1028            if (layoutControlItemchartControlSplit5.Visible)
1029            {
1030              splitterItem5and7.Visibility = LayoutVisibility.Always;
1031            }
1032            if (layoutControlItemchartControlSplit6.Visible)
1033            {
1034              splitterItem6and5.Visibility = LayoutVisibility.Always;

1035            }
1036            meter.IsShown = true;
1037          }
1038          break;
1039        case 8:
1040          if (meter.IsShown)
1041          {
1042            layoutControlItemchartControlSplit8.Visibility = LayoutVisibility.
                Never;
1043            if (!layoutControlItemchartControlSplit1.Visible)
1044            {
1045              splitterItem1And8.Visibility = LayoutVisibility.Never;
1046            }
1047            if (!layoutControlItemchartControlSplit4.Visible)
1048            {
1049              splitterItem8and4.Visibility = LayoutVisibility.Never;
1050            }
1051            meter.IsShown = false;
1052          }
1053          else
1054          {
1055            layoutControlItemchartControlSplit8.Visibility = LayoutVisibility.
                Always;
1056            if (layoutControlItemchartControlSplit1.Visible)
1057            {
1058              splitterItem1And8.Visibility = LayoutVisibility.Always;
1059            }
1060            if (layoutControlItemchartControlSplit4.Visible)
1061            {
1062              splitterItem8and4.Visibility = LayoutVisibility.Always;
1063            }
1064            meter.IsShown = true;
1065          }
1066          break;
1067        default:
1068          break;
1069      }
1070    }
```

```
1071    private void btnSingleMeterStartLiveReadings_ItemClick(object sender,
          ItemClickEventArgs e)
1072    {
```

```
1073        Cursor.Current = Cursors.WaitCursor;
1074        SetUpRepositorys();
1075        LoadMeters();
1076        SetConstantEnergyConstantLineForChart(chartControlEnergyReadings);
1077        SePeakPowerConstantLineForLivePowerChart(chartControlPowerReadings);

1078        Meter m = repositoryItemGridLookUpEditAllMetersItem.EditValue as Meter;
1079        if (m != null)
1080        {
1081          isFirstTimeSingleMeter = true;
1082          gmap.Position = Location(m.id);
1083          gmap.Zoom = 19;
1084          Series series = chartControlPowerReadings.Series[0];
1085          Series energySeries = chartControlEnergyReadings.Series[0];
1086          series.Points.Clear();//.RemoveRange(0, series.Points.Count);
1087          energySeries.Points.Clear();
1088          initialTimeInterval = 60;
1089          initialEnergyTimeInterval = 600;
1090          meterID = m.id;
1091          shortTimeInterval = 5;
1092          LoadAllSingleMeterReadings(initialTimeInterval,
                initialEnergyTimeInterval, meterID, shortTimeInterval);
1093        }
1094        timer1SingleMeter.Enabled = true;
1095        timerMultiMeterScreens.Enabled = false;//just making sure we dont have to
              many timer running
1096        dockPanelSingleMeterLiveEnergyReadings.DockAsMdiDocument();
1097        dockPanelSingleMeterLivePoower.DockAsMdiDocument();
1098        Cursor.Current = Cursors.Default;
1099      }
```

```
1100      /// <summary>
1101      /// Method to set the constan line for the energy chart.
1102      /// constant line is the peak value line.
1103      /// </summary>
1104      /// <param name="chartControlEnergyReadings"></param>
1105      private void SetConstantEnergyConstantLineForChart(ChartControl
            chartControlEnergyReadings)
1106      {
1107        SwiftPlotDiagram diagram = (SwiftPlotDiagram)chartControlEnergyReadings.
            Diagram;
1108        diagram.AxisY.ConstantLines[0].AxisValue = spnEditPeakEnergyLine.EditValu
            ;
1109      }
```

```
1110      /// <summary>
1111      /// method to set peak power constant line for live power chart
1112      /// </summary>
1113      /// <param name="chartControlPowerReadings"></param>
1114      private void SePeakPowerConstantLineForLivePowerChart(ChartControl
            chartControlPowerReadings)
1115      {
1116        SwiftPlotDiagram diagram = (SwiftPlotDiagram)chartControlPowerReadings.
            Diagram;
```

```
1117        diagram.AxisY.ConstantLines[0].AxisValue = spnEditPeakPowerValue.EditValu
              ;
1118      }


1119      /// <summary>
1120      /// Method to set up the available chart types
1121      /// </summary>
1122      private void SetUpChartTypes()
1123      {
1124        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Bar);
1125        repositoryItemComboBoxChartTypes.Items.Add(ViewType.SideBySideStackedBar3
            );
1126        repositoryItemComboBoxChartTypes.Items.Add(ViewType.FullStackedBar);
1127        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Pie3D);
1128        repositoryItemComboBoxChartTypes.Items.Add(ViewType.FullStackedLine3D);
1129        repositoryItemComboBoxChartTypes.Items.Add(ViewType.FullStackedLine);
1130        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Pie);
1131        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Area3D);
1132        repositoryItemComboBoxChartTypes.Items.Add(ViewType.ManhattanBar);
1133        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Area3D);
1134        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Bar3D);
1135        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Funnel3D);
1136        repositoryItemComboBoxChartTypes.Items.Add(ViewType.FullStackedArea3D);
1137        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Line3D);
1138        repositoryItemComboBoxChartTypes.Items.Add(ViewType.StackedArea3D);
1139        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Bubble);
1140        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Doughnut);
1141        repositoryItemComboBoxChartTypes.Items.Add(ViewType.SideBySideStackedBar3
            );
1142        repositoryItemComboBoxChartTypes.Items.Add(ViewType.StackedLine);
1143        repositoryItemComboBoxChartTypes.Items.Add(ViewType.SideBySideStackedBar);
1144        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Spline3D);
1145        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Spline);
1146        repositoryItemComboBoxChartTypes.Items.Add(ViewType.SplineArea3D);
1147        repositoryItemComboBoxChartTypes.Items.Add(ViewType.StackedSplineArea);
1148        repositoryItemComboBoxChartTypes.Items.Add(ViewType.StackedSplineArea3D);
1149        repositoryItemComboBoxChartTypes.Items.Add(ViewType.SwiftPlot);
1150        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Line);
1151        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Point);

1152        comboBoxEdtChartTypes.EditValue = ViewType.Bar3D;
1153      }


1154      private void comboBoxEdtChartTypes_EditValueChanged(object sender, EventArg
            e)
1155      {
1156        // comboBoxEdtChartTypes.EditValue
1157        //      chartControlSplit1.Series[0].ChangeView(ViewType.SwiftPlot);
1158        //
      ucPivotGridChartControl.ChartControl.SeriesTemplate.ChangeView((ViewType)((itmcbChartView


1159        chartControlMeterEnergyArchive.SeriesTemplate.ChangeView((ViewType)((
              comboBoxEdtChartTypes.EditValue)));
```

```
1160        chartControlMeterPowerArchive.SeriesTemplate.ChangeView((ViewType)((
              comboBoxEdtChartTypes.EditValue)));
1161        chartControlAllMetersEnergyWindowArchive.SeriesTemplate.ChangeView((
              ViewType)((comboBoxEdtChartTypes.EditValue)));
1162        Diagram3D diagram1 = chartControlMeterEnergyArchive.Diagram as Diagram3D;
1163        if (diagram1 != null)
1164        {
1165          diagram1.RuntimeRotation = true;
1166          diagram1.RuntimeScrolling = true;
1167          diagram1.RuntimeZooming = true;

1168        }
1169        else
1170        {
1171          XYDiagram diagram11 = chartControlMeterEnergyArchive.Diagram as
                XYDiagram;
1172          if (diagram11 != null)
1173          {
1174            diagram11.EnableAxisXScrolling = true;
1175            diagram11.EnableAxisXZooming = true;
1176            diagram11.EnableAxisYScrolling = true;
1177            diagram11.EnableAxisYZooming = true;
1178          }
1179          else
1180          {
1181            XYDiagram3D diagram3d = chartControlMeterEnergyArchive.Diagram as
                  XYDiagram3D;
1182            if (diagram3d != null)
1183            {
1184              diagram3d.RuntimeScrolling = true;
1185              diagram3d.RuntimeZooming = true;
1186              diagram3d.RuntimeRotation = true;
1187            }
1188            else
1189            {

1190              SwiftPlotDiagram diagramsp = chartControlMeterEnergyArchive.Diagram
                    as SwiftPlotDiagram;
1191              if (diagramsp != null)
1192              {
1193                diagramsp.EnableAxisXScrolling = true;
1194                //  diagramsp.EnableAxisYScrolling = true;
1195                diagramsp.EnableAxisXZooming = true;

1196                diagramsp.AxisX.NumericOptions.Format = DevExpress.XtraCharts.
                      NumericFormat.General;
1197                diagramsp.AxisX.Range.ScrollingRange.SideMarginsEnabled = true;
1198                diagramsp.AxisX.Range.SideMarginsEnabled = true;
1199                diagramsp.AxisX.DateTimeGridAlignment = DevExpress.XtraCharts.
                      DateTimeMeasurementUnit.Minute;
1200                diagramsp.AxisX.DateTimeMeasureUnit = DevExpress.XtraCharts.
                      DateTimeMeasurementUnit.Minute;
1201                diagramsp.AxisX.DateTimeOptions.Format = DevExpress.XtraCharts.
                      DateTimeFormat.Custom;
1202                diagramsp.AxisX.DateTimeOptions.FormatString = "MMM d HH:mm:ss";
1203                diagramsp.AxisX.GridLines.Visible = true;
```

```
1204                diagramsp.AxisX.NumericOptions.Format = DevExpress.XtraCharts.
                      NumericFormat.General;
1205                diagramsp.AxisX.Range.ScrollingRange.SideMarginsEnabled = true;
1206                diagramsp.AxisX.Range.SideMarginsEnabled = true;
1207                diagramsp.AxisX.Tickmarks.CrossAxis = true;
1208                diagramsp.AxisX.VisibleInPanesSerializable = "-1";

1209            }

1210        }
1211    }
1212}


1213    Diagram3D diagram2 = chartControlMeterPowerArchive.Diagram as Diagram3D;
1214    if (diagram2 != null)
1215    {
1216      diagram2.RuntimeRotation = true;
1217      diagram2.RuntimeScrolling = true;
1218      diagram2.RuntimeZooming = true;

1219    }
1220    else
1221    {
1222      XYDiagram diagram211 = chartControlMeterPowerArchive.Diagram as
            XYDiagram;
1223      if (diagram211 != null)
1224      {
1225        diagram211.EnableAxisXScrolling = true;
1226        diagram211.EnableAxisXZooming = true;
1227        diagram211.EnableAxisYScrolling = true;
1228        diagram211.EnableAxisYZooming = true;
1229      }

1230      else
1231      {
1232        XYDiagram3D diagram3d = chartControlMeterEnergyArchive.Diagram as
              XYDiagram3D;
1233        if (diagram3d != null)
1234        {
1235          diagram3d.RuntimeScrolling = true;
1236          diagram3d.RuntimeZooming = true;
1237          diagram3d.RuntimeRotation = true;

1238        }
1239      }
1240    }
1241    Diagram3D diagram3 = chartControlAllMetersEnergyWindowArchive.Diagram as
        Diagram3D;
1242    if (diagram3 != null)
1243    {
1244      diagram3.RuntimeRotation = true;
1245      diagram3.RuntimeScrolling = true;
1246      diagram3.RuntimeZooming = true;

1247    }
```

```
1248        else
1249        {
1250          XYDiagram diagram311 = chartControlAllMetersEnergyWindowArchive.Diagram
              as XYDiagram;
1251          if (diagram311 != null)
1252          {
1253            diagram311.EnableAxisXScrolling = true;
1254            diagram311.EnableAxisXZooming = true;
1255            diagram311.EnableAxisYScrolling = true;
1256            diagram311.EnableAxisYZooming = true;
1257          }
1258          else
1259          {
1260            XYDiagram3D diagram3d = chartControlMeterEnergyArchive.Diagram as
                XYDiagram3D;
1261            if (diagram3d != null)
1262            {
1263              diagram3d.RuntimeScrolling = true;
1264              diagram3d.RuntimeZooming = true;
1265              diagram3d.RuntimeRotation = true;
1266            }
1267          }
1268        }
1269        ViewType vt = ((ViewType)((comboBoxEdtChartTypes.EditValue)));
1270        if (vt == ViewType.SwiftPlot)
1271        {
1272          List<EnergyReading> readings = pivotGridControlSingleMeterEnergyArchive
              DataSource as List<EnergyReading>;
1273          //

1274          if (readings != null)
1275          {
1276            Series seriesEnergy = chartControlMeterEnergyArchive.Series[0];
1277            SeriesPoint[] pointsToUpdate = new SeriesPoint[readings.Count];
1278            for (int i = 0; i < readings.Count; i++)
1279            {
1280              EnergyReading r = readings[i];
1281              pointsToUpdate[i] = new SeriesPoint(r.measurement_timestamp, r.valu
                    );


1282            }
1283            seriesEnergy.Points.AddRange(pointsToUpdate);
1284          }
1285          //



1286        }
1287      }
```

```
1288      /// <summary>
1289      /// Button to load archived enery readings for all the meters.
1290      /// </summary>
1291      /// <param name="sender"></param>
1292      /// <param name="e"></param>
1293      private void btnLaunchMultiMeterEnergyReadingsArchive_ItemClick(object
```

```
          sender, ItemClickEventArgs e)
1294      {
1295        Cursor.Current = Cursors.WaitCursor;
1296        if (meterRepository.allMeterDetails != null)
1297        {
1298          pivotGridControlSingleMeterEnergyArchive.DataSource = null;
1299          chartControlMeterEnergyArchive.DataSource = null;
1300          gridControlRawEnergyReadings.DataSource = null;
1301          //  pivotGridControlSingleMeterEnergyArchive.CollapseAll();
1302          pivotGridControlSingleMeterEnergyArchive.CollapseAll();
1303          //  pivotGridControlAllMetersArchive.CollapseAll();
1304          energyReadingRepository.MeterDetails = meterRepository.allMeterDetails;

1305          DateTime dateTimeFromThisDat = (DateTime)
                dateTimeMultiMeterTimeFromThisDate.EditValue;
1306          DateTime dateTimeToThisDat = (DateTime)dateTimeMultiMeterTimeToThisDate
                EditValue;
1307          List<EnergyReading> dings = energyReadingRepository.
                LoadEnergyReadingsBasedOnDates(dateTimeFromThisDat, dateTimeToThisDat
                 (double)spnEditorEnergyPricePerKwhr.EditValue);

1308          if (dings.Count > 0)
1309          {

1310            pivotGridControlSingleMeterEnergyArchive.DataSource = dings;
1311            chartControlMeterEnergyArchive.DataSource =
                  pivotGridControlSingleMeterEnergyArchive;
1312            gridControlRawEnergyReadings.DataSource = dings;
1313            documentManager.BeginUpdate();
1314            if (!dockPanelRawEnergyReadings.IsMdiDocument)
1315            {
1316              dockPanelRawEnergyReadings.DockAsMdiDocument();
1317            }
1318            if (!dockPanelMeterEnergyReadingsArchive.IsMdiDocument)
1319            {
1320              dockPanelMeterEnergyReadingsArchive.DockAsMdiDocument();
1321            }
1322            documentManager.EndUpdate();
1323          }
1324          else
1325          {

1326            MessageBox.Show("There are no archived readings in the selected time
                  range!", "Wits Dasboard",
1327            MessageBoxButtons.OK, MessageBoxIcon.Error);
1328          }

1329        }

1330        Cursor.Current = Cursors.Default;
1331      }


1332      /// <summary>
1333      /// Handler to launch the weather info browser
1334      /// </summary>
```

```
1335        /// <param name="sender"></param>
1336        /// <param name="e"></param>
1337        private void btnLaunchWeatherCenter_ItemClick(object sender,
              ItemClickEventArgs e)
1338        {
1339          webBrowserWeather.Navigate("http://www.weathersa.co.za/web/index.php");
1340          if (!dockPanelWeatherCenter.IsMdiDocument)
1341          {
1342            dockPanelWeatherCenter.DockAsMdiDocument();
1343          }
1344        }

1345      }

1346      /// <summary>
1347      /// class to model the main form for the app
1348      /// </summary>
1349      public partial class CopyOfFrmMain : DevExpress.XtraBars.Ribbon.RibbonForm
1350      {
1351        /// <summary>
1352        /// file reader
1353        /// </summary>
1354        private BaselineFileReader FileReader;
1355        /// <summary>
1356        /// instance of calculator control
1357        /// </summary>
1358        private AlgorithmCalculations AlgorithmCalculator;
1359        private MeterRepository meterRepository;
1360        private PowerReadingRepository readingRepository;
1361        private EnergyReadingRepository energyReadingRepository;
1362        private EnergyWindowsRepository energyWindowsRepository;
1363        private List<Meter> Meters;
1364        private List<PowerReading> Readings;
1365        private List<DbEnergyReading> EnergyReadings;
1366        private bool isFirstTimeSingleMeter = true;
1367        private bool isFirstTimeMultiMeter = true;
1368        private int initialTimeInterval = 600;
1369        private int initialEnergyTimeInterval = 600;
1370        private int meterID = 1;
1371        private int shortTimeInterval = 5;
1372        public bool hasForecastBeenGivenForThisHour = false;
1373        private List<ChartControl> SplitScreenChartControls = new List<ChartControl
              ();
1374        private bool isPreviousPeackAvailable = false;
1375        private NpgsqlConnection conn;

1376        public float PreviousPeak {
1377        }

1378        public CopyOfFrmMain()
1379        {
1380          InitializeComponent();
1381        }

1382        private void FrmMain_Load(object sender, EventArgs e)
```

```
1383      {
1384        DevExpress.LookAndFeel.UserLookAndFeel.Default.SkinName = "Caramel";
1385      }

1386      /// <summary>
1387      /// Method that sets up all main form data
1388      /// </summary>
1389      public void LoadMainForm(ConfigSettings cS)
1390      {
1391        FileReader = new BaselineFileReader();

1392        List<BaselineVariable> list = FileReader.GetVariables();
1393        AlgorithmCalculator = new AlgorithmCalculations(list);
1394        CreateConnection(cS);
1395        SetUpRepositorys();
1396        SetUpMultiMeterArchiveUI();
1397        dockPanelSingleMeterLivePoower.Dock = DockingStyle.Fill;
1398        SetUpChartTypes();
1399        try
1400        {
1401          LoadMeters();
1402        }
1403        catch (Exception ex)
1404        {
1405          MessageBox.Show("No Available Database connection:\n" + ex.ToString());
1406        }
1407      }

1408      private void CreateConnection(ConfigSettings cS)
1409      {
1410        NpgsqlConnectionStringBuilder csb = new NpgsqlConnectionStringBuilder();
1411        csb.Database = cS.DatabaseName;
1412        csb.Host = cS.ServerName;
1413        csb.UserName = cS.DatabaseUserName; // must match Common Name of client
      certificate
1414        csb.Password = cS.Password;
1415        csb.SSL = true;
1416        csb.Port = cS.PortNumber;
1417        csb.SslMode = SslMode.Require;

1418        conn = new NpgsqlConnection(csb.ConnectionString);
1419      }

1420      private void SetUpRepositorys()
1421      {
1422        Database.SetInitializer<MeterRepository>(null);
1423        Database.SetInitializer<PowerReadingRepository>(null);
1424        Database.SetInitializer<EnergyReadingRepository>(null);
1425        Database.SetInitializer<EnergyWindowsRepository>(null);

1426        meterRepository = new MeterRepository(conn);
1427        readingRepository = new PowerReadingRepository(conn);
1428        energyReadingRepository = new EnergyReadingRepository(conn);
```

```
1429        energyWindowsRepository = new EnergyWindowsRepository(conn);
1430        Meters = new List<Meter>();
1431        Readings = new List<PowerReading>();
1432        EnergyReadings = new List<DbEnergyReading>();
1433        // repositoryMeterItemLookUpEdit.DataSource = Meters;
1434        repositoryItemGridLookUpEditAllMeters.DataSource = Meters;
1435        // gridControlPowerReadings.DataSource = Readings;

1436      }


1437      private void LoadMeters()
1438      {
1439        //try
1440        //    {

1441        Meters = meterRepository.LoadMeters();
1442        repositoryMeterItemLookUpEdit.DataSource = Meters;
1443        repositoryItemGridLookUpEditAllMeters.DataSource = Meters;
1444        energyWindowsRepository.meters = meterRepository.allMeterDetails;
1445        // }
1446        //   catch (Npgsql.NpgsqlException e)
1447        // {

1448        //   MessageBox.Show("No Database connectivity"+"-"+e.Message.ToString());
1449        //  }

1450      }


1451      private void LoadReadings()
1452      {
1453        readingRepository.LoadReadings(4, 1);
1454        Readings = readingRepository.LoadMonthlyReadings(90, 0, 0, 1);
1455        gridControlPowerReadings.DataSource = Readings;
1456      }


1457      /// <summary>
1458      /// Method that loads the map provider
1459      /// </summary>
1460      private void LoadGoogleMap()
1461      {
1462        gmap.MapProvider = GMap.NET.MapProviders.GoogleMapProvider.Instance;
1463        GMap.NET.GMaps.Instance.Mode = GMap.NET.AccessMode.ServerOnly;
1464        gmap.Position = new PointLatLng(-26.191693, 28.026967);
1465        gmap.Zoom = 18;
1466      }


1467      private PointLatLng Location(int locationID)
1468      {
1469        PointLatLng point = new PointLatLng(-26.191693, 28.026967);
1470        String path = "";
1471        switch (locationID)
1472        {
```

```
1473            case 1:
1474                point = new PointLatLng(-26.19181, 28.026123);
1475                path = "raikesRoad.jpg";
1476                AddCustomMarkerToMap(point, path);
1477                break;
1478            case 2:
1479                point = new PointLatLng(-26.191719, 28.031968);
1480                path = "GateHouse.jpg";
1481                AddCustomMarkerToMap(point, path);
1482                break;
1483            case 3:
1484                point = new PointLatLng(-26.177383, 28.046684);
1485                path = "medicalSchool.jpg";
1486                AddCustomMarkerToMap(point, path);
1487                break;
1488            case 4:
1489                point = new PointLatLng(-26.177383, 28.046684);
1490                path = "medicalSchool.jpg";
1491                AddCustomMarkerToMap(point, path);
1492                break;
1493            case 5:
1494                point = new PointLatLng(-26.177383, 28.046684);
1495                path = "medicalSchool.jpg";
1496                AddCustomMarkerToMap(point, path);
1497                break;
1498            case 6:
1499                point = new PointLatLng(-26.178399, 28.04206);
1500                path = "WECEducationCampus.jpg";
1501                AddCustomMarkerToMap(point, path);
1502                break;
1503            case 7:
1504                point = new PointLatLng(-26.179901, 28.037543);
1505                path = "EOHresPKV.jpg";
1506                AddCustomMarkerToMap(point, path);
1507                break;
1508            case 8:
1509                point = new PointLatLng(-26.191825, 28.031985);
1510                path = "commerceLibrary.jpg";
1511                AddCustomMarkerToMap(point, path);
1512                break;
1513            case 9://Gemni lab...put correct icon
1514                point = new PointLatLng(-26.191719, 28.031968);
1515                path = "commerceLibrary.jpg";
1516                AddCustomMarkerToMap(point, path);
1517                break;
1518            default:
1519                point = new PointLatLng(-26.189408, 28.026053);
1520                path = "raikesRoad.jpg";
1521                AddCustomMarkerToMap(point, path);
1522                break;
1523        }
1524        return point;
1525    }
```

```
1526    private void AddCustomMarkerToMap(PointLatLng point, string path)
```

```
1527      {
1528        gmap.Overlays.Clear();
1529        GMapOverlay top = new GMapOverlay(gmap, "top");
1530        gmap.Overlays.Add(top);
1531        Image marker = Image.FromFile(path);
1532        GMapCustomMarker gCm = new GMapCustomMarker(point, marker);
1533        top.Markers.Add(gCm);
1534      }

1535      private void btnLauchGISMap_ItemClick(object sender, ItemClickEventArgs e)
1536      {
1537        if (!dockPanelMap.IsMdiDocument)
1538        {
1539          LoadGoogleMap();
1540          dockPanelMap.DockAsMdiDocument();
1541          // AddCustomMarkerToMap();
1542        }
1543      }

1544      private void AddCustomMarkerToMap()
1545      {
1546        GMapOverlay top = new GMapOverlay(gmap, "top");
1547        gmap.Overlays.Add(top);
1548        //gmap.Overlays[0].Markers.Add(
1549        PointLatLng position = new PointLatLng(-26.191693, 28.026967);
1550        Image test = Image.FromFile("Dependencies\\launch6.jpg");
1551        GMapCustomMarker ddasd = new GMapCustomMarker(position, test);
1552        top.Markers.Add(ddasd);
1553      }

1554      /// <summary>
1555      /// Method to load all single meter readings
1556      /// </summary>
1557      /// <param name="initialTimeInterval"></param>
1558      /// <param name="initialEnergyTimeInterval"></param>
1559      /// <param name="meterID"></param>
1560      /// <param name="shortTimeInterval"></param>
1561      private void LoadAllSingleMeterReadings(int initialTimeInterval, int
          initialEnergyTimeInterval, int meterID, int shortTimeInterval)
1562      {
1563        Series series = chartControlPowerReadings.Series[0];
1564        if (series == null)
1565          return;
1566        Series seriesEnergy = chartControlEnergyReadings.Series[0];

1567        if (seriesEnergy == null)
1568          return;
1569        List<DbPowerReading> currentPowerReadings = null;
1570        List<DbEnergyReading> currentEnergyReadings = null;
1571        if (isFirstTimeSingleMeter)
1572        {
1573          currentPowerReadings = RefreshSingleMeterPowerReadings(
              initialTimeInterval, meterID, currentPowerReadings);
```

```
1574            currentEnergyReadings = RefreshSingleMeterEnergyReadings(
                  initialEnergyTimeInterval, meterID, currentEnergyReadings);
1575            GetPrevioiusPeakForMeter(meterID);
1576          }
1577          else
1578          {

1579            currentPowerReadings = RefreshSingleMeterPowerReadings(shortTimeInterval
                  , meterID, currentPowerReadings);
1580            currentEnergyReadings = RefreshSingleMeterEnergyReadings(
                  shortTimeInterval, meterID, currentEnergyReadings);
1581            WarnPeakEnergyApproaching(currentEnergyReadings);
1582          }
1583          if (currentPowerReadings != null)
1584          {
1585            SeriesPoint[] pointsToUpdate = new SeriesPoint[currentPowerReadings.
                  Count];
1586            for (int i = 0; i < currentPowerReadings.Count; i++)
1587            {
1588              DbPowerReading r = currentPowerReadings[i];
1589              pointsToUpdate[i] = new SeriesPoint(r.metertimestamp, r.powertotal);

1590            }
1591            series.Points.AddRange(pointsToUpdate);
1592          }
1593          if (currentEnergyReadings != null)
1594          {
1595            SeriesPoint[] pointsToUpdate = new SeriesPoint[currentEnergyReadings.
                  Count];
1596            for (int i = 0; i < currentEnergyReadings.Count; i++)
1597            {
1598              DbEnergyReading r = currentEnergyReadings[i];
1599              pointsToUpdate[i] = new SeriesPoint(r.measurement_timestamp, r.value);

1600            }
1601            seriesEnergy.Points.AddRange(pointsToUpdate);
1602          }
1603        }
```

```
1604        /// <summary>
1605        /// Method to check if any of the values are approachig peak and give a
              warning.
1606        /// </summary>
1607        /// <param name="currentEnergyReadings"></param>
1608        private void WarnPeakEnergyApproaching(List<DbEnergyReading>
              currentEnergyReadings)
1609        {
1610          int percentage = Convert.ToInt32(spnEditPercentageToPeak.EditValue);
1611          double fraction = (double)percentage / (100);
1612          int peakValue = Convert.ToInt32(spnEditPeakEnergyLine.EditValue);
1613          double warn = peakValue - (fraction * peakValue);
1614          int percentage1 = Convert.ToInt32(spnEditPercentageToPeak.EditValue);
1615          if (currentEnergyReadings.Any(x => x.value >= warn))
1616          {
1617            MessageBox.Show("The energy value is within \"" + percentage1.ToString(
```

```
                + "%\"of set limit value!", "Wits Dasboard",
1618            MessageBoxButtons.OK, MessageBoxIcon.Warning);
1619        }

1620        DateTime currentTime = DateTime.Now;

1621        WebRequest req = HttpWebRequest.Create("
              http://api.openweathermap.org/data/2.5/weather?q=Johannesburg&mode=xml&units=metri
              ");
1622        WebResponse res = req.GetResponse();
1623        StreamReader sr = new StreamReader(res.GetResponseStream());
1624        string str = sr.ReadToEnd();
1625        XmlDocument doc = new XmlDocument();
1626        doc.LoadXml(str);
1627        XmlNodeList parentNode = doc.GetElementsByTagName("current");
1628        string temperetureValue = "";
1629        foreach (XmlNode item in parentNode)
1630        {
1631            temperetureValue = item.SelectSingleNode("temperature").Attributes.
                GetNamedItem("value").InnerText;// ("value").InnerText;
1632        }
1633        float tempValue = 0;
1634        if (float.TryParse(temperetureValue, out tempValue) &&
              isPreviousPeackAvailable)
1635        {

1636            float forecastLoad = AlgorithmCalculator.getBaselineValue((currentTime.
                Hour) * 2, eCampusStatus.Special, currentTime.DayOfWeek, currentTime.
                Month);
1637            double forecast = AlgorithmCalculator.CalculateForecastLoadForCycle(
                tempValue, PreviousPeak, forecastLoad);
1638            MessageBox.Show("The forecast peak load in this cycle is  \"" + forecas
                .ToString() + "\"kWh", "Wits Dasboard",
1639            MessageBoxButtons.OK, MessageBoxIcon.Warning);
1640        }
1641    }
```

```
1642    /// <summary>
1643    /// Method to setu up split screen chart controls
1644    /// </summary>
1645    private void SetUpSplitScreenChartControls()
1646    {
1647        SplitScreenChartControls.Add(chartControlSplit1);
1648        SplitScreenChartControls.Add(chartControlSplit2);
1649        SplitScreenChartControls.Add(chartControlSplit3);
1650        SplitScreenChartControls.Add(chartControlSplit4);
1651    }
```

```
1652    private void LoadAllMultiMeterReadings(int initialTimeInterval, int
          initialEnergyTimeInterval, int shortTimeInterval)
1653    {
1654        Series series = chartControlPowerReadings.Series[0];
1655        if (series == null)
1656            return;
```

```
1657        Series seriesEnergy1 = chartControlSplit1.Series[0];
1658        Series seriesEnergy2 = chartControlSplit2.Series[0];
1659        Series seriesEnergy3 = chartControlSplit3.Series[0];
1660        Series seriesEnergy4 = chartControlSplit4.Series[0];
1661        Series seriesEnergy5 = chartControlSplit5.Series[0];
1662        Series seriesEnergy6 = chartControlSplit6.Series[0];
1663        Series seriesEnergy7 = chartControlSplit7.Series[0];
1664        Series seriesEnergy8 = chartControlSplit8.Series[0];
1665        List<DbPowerReading> currentPowerReadings = null;
1666        List<DbEnergyReading> currentEnergyReadings = null;
1667        if (isFirstTimeMultiMeter)
1668        {
1669          currentPowerReadings = RefreshMultimeterPowerReadings(
              initialTimeInterval, currentPowerReadings);
1670          currentEnergyReadings = RefreshMultiMeterEnergyReadings(
              initialEnergyTimeInterval, currentEnergyReadings);

1671        }
1672        else
1673        {

1674          currentPowerReadings = RefreshMultimeterPowerReadings(shortTimeInterval
              currentPowerReadings);//
      readingRepository.LoadReadings(shortTimeInterval, meterID);
1675          currentEnergyReadings = RefreshMultiMeterEnergyReadings(
              shortTimeInterval, currentEnergyReadings);//
      energyReadingRepository.LoadReadings(shortTimeInterval, meterID);
1676        }

1677        //ToDo rodwell... Get distinct meter IDs...create energy reading lists fo
              each meter. Populate a chart control for each list
1678        //hide unpopulated charts.If list of Ids is greater than the initial 4
      then create a chart control on the fly.add meter ID to the chartcontrol
1679        //tag so that when u refresh u know which chart a reading belongs to.


1680        List<DbEnergyReading> one = new List<DbEnergyReading>();//put
      instantiation in the first time bool...so that u dont delete vales on
      refress..or is it important since we dont clear actuall series?
1681        List<DbEnergyReading> two = new List<DbEnergyReading>();
1682        List<DbEnergyReading> three = new List<DbEnergyReading>();
1683        List<DbEnergyReading> four = new List<DbEnergyReading>();
1684        List<DbEnergyReading> five = new List<DbEnergyReading>();
1685        List<DbEnergyReading> six = new List<DbEnergyReading>();
1686        List<DbEnergyReading> seven = new List<DbEnergyReading>();
1687        List<DbEnergyReading> eight = new List<DbEnergyReading>();

1688        foreach (DbEnergyReading item in currentEnergyReadings)
1689        {
1690          switch (item.meter_id)
1691          {
1692            case 1:
1693              one.Add(item);
1694              break;
1695            case 2:
1696              two.Add(item);
```

```
1697                    break;
1698                case 3:
1699                    three.Add(item);
1700                    break;
1701                case 4:
1702                    four.Add(item);
1703                    break;
1704                case 5:
1705                    five.Add(item);
1706                    break;
1707                case 6:
1708                    six.Add(item);
1709                    break;
1710                case 7:
1711                    seven.Add(item);
1712                    break;
1713                case 8:
1714                    eight.Add(item);
1715                    break;
1716                default:
1717                    break;
1718            }

1719        }
1720        //if (currentPowerReadings != null)
1721        //{
1722        //  SeriesPoint[] pointsToUpdate = new
    SeriesPoint[currentPowerReadings.Count];
1723        //  for (int i = 0; i < currentPowerReadings.Count; i++)
1724        //  {
1725        //    DbPowerReading r = currentPowerReadings[i];
1726        //    pointsToUpdate[i] = new SeriesPoint(r.metertimestamp, r.powertotal);

1727        //  }
1728        //  series.Points.AddRange(pointsToUpdate);
1729        //}

1730        if (isFirstTimeMultiMeter)
1731        {
1732            if (one.Count == 0)
1733            {
1734                layoutControlItemchartControlSplit1.Visibility = LayoutVisibility.
                      Never;
1735                splitterItem1And8.Visibility = LayoutVisibility.Never;
1736                Meters.FirstOrDefault(x => x.id == 1).IsShown = false;
1737            }
1738            if (two.Count == 0)
1739            {
1740                layoutControlItemchartControlSplit2.Visibility = LayoutVisibility.
                      Never;
1741                splitterItem7and2.Visibility = LayoutVisibility.Never;
1742                Meters.FirstOrDefault(x => x.id == 2).IsShown = false;
1743            }
1744            if (three.Count == 0)
1745            {
1746                layoutControlItemchartControlSplit3.Visibility = LayoutVisibility.
```

```
            Never;
1747        splitterItem4and3.Visibility = LayoutVisibility.Never;
1748        Meters.FirstOrDefault(x => x.id == 3).IsShown = false;

1749      }
1750      if (four.Count == 0)
1751      {
1752        layoutControlItemchartControlSplit4.Visibility = LayoutVisibility.
            Never;
1753        if (!layoutControlItemchartControlSplit3.Visible)
1754        {
1755          splitterItem4and3.Visibility = LayoutVisibility.Never;
1756        }
1757        if (!layoutControlItemchartControlSplit8.Visible)
1758        {
1759          splitterItem8and4.Visibility = LayoutVisibility.Never;
1760        }

1761        Meters.FirstOrDefault(x => x.id == 4).IsShown = false;
1762      }
1763      if (five.Count == 0)
1764      {
1765        layoutControlItemchartControlSplit5.Visibility = LayoutVisibility.
            Never;
1766        if (!layoutControlItemchartControlSplit7.Visible)
1767        {
1768          splitterItem5and7.Visibility = LayoutVisibility.Never;
1769        }
1770        if (!layoutControlItemchartControlSplit6.Visible)
1771        {
1772          splitterItem6and5.Visibility = LayoutVisibility.Never;
1773        }

1774        Meters.FirstOrDefault(x => x.id == 5).IsShown = false;
1775      }
1776      if (six.Count == 0)
1777      {
1778        layoutControlItemchartControlSplit6.Visibility = LayoutVisibility.
            Never;
1779        splitterItem6and5.Visibility = LayoutVisibility.Never;
1780        Meters.FirstOrDefault(x => x.id == 6).IsShown = false;
1781      }
1782      if (seven.Count == 0)
1783      {
1784        layoutControlItemchartControlSplit7.Visibility = LayoutVisibility.
            Never;
1785        if (!layoutControlItemchartControlSplit5.Visible)
1786        {
1787          splitterItem5and7.Visibility = LayoutVisibility.Never;
1788        }
1789        if (!layoutControlItemchartControlSplit2.Visible)
1790        {
1791          splitterItem7and2.Visibility = LayoutVisibility.Never;
1792        }

1793        Meters.FirstOrDefault(x => x.id == 7).IsShown = false;
```

```
1794            }
1795            if (eight.Count == 0)
1796            {
1797              layoutControlItemchartControlSplit8.Visibility = LayoutVisibility.
                    Never;
1798              if (!layoutControlItemchartControlSplit1.Visible)
1799              {
1800                splitterItem1And8.Visibility = LayoutVisibility.Never;
1801              }
1802              if (!layoutControlItemchartControlSplit4.Visible)
1803              {
1804                splitterItem8and4.Visibility = LayoutVisibility.Never;
1805              }

1806              //  Meters.FirstOrDefault(x => x.id == 8).IsShown = false;
1807            }
1808          }

1809          AddMultiEnergyValuesToChatSeries(seriesEnergy1, one);
1810          AddMultiEnergyValuesToChatSeries(seriesEnergy2, two);
1811          AddMultiEnergyValuesToChatSeries(seriesEnergy3, three);
1812          AddMultiEnergyValuesToChatSeries(seriesEnergy4, four);
1813          AddMultiEnergyValuesToChatSeries(seriesEnergy5, five);
1814          AddMultiEnergyValuesToChatSeries(seriesEnergy6, six);
1815          AddMultiEnergyValuesToChatSeries(seriesEnergy7, seven);
1816          AddMultiEnergyValuesToChatSeries(seriesEnergy8, eight);
1817      }
```

```
1818      /// <summary>
1819      /// method to add multi energy values to chart series
1820      /// </summary>
1821      /// <param name="seriesEnergy1"></param>
1822      /// <param name="one"></param>
1823      private static void AddMultiEnergyValuesToChatSeries(Series seriesEnergy1,
            List<DbEnergyReading> one)
1824      {
1825          if (one.Count > 0)
1826          {
1827              SeriesPoint[] pointsToUpdate = new SeriesPoint[one.Count];
1828              for (int i = 0; i < one.Count; i++)
1829              {
1830                  DbEnergyReading r = one[i];
1831                  pointsToUpdate[i] = new SeriesPoint(r.measurement_timestamp, r.value);

1832              }
1833              seriesEnergy1.Points.AddRange(pointsToUpdate);
1834          }
1835      }
```

```
1836      private List<DbEnergyReading> RefreshSingleMeterEnergyReadings(int
            initialTimeInterval, int meterID, List<DbEnergyReading>
            currentEnergyReadings)
1837      {
1838          timer1SingleMeter.Stop();
```

```
1839          currentEnergyReadings = energyReadingRepository.LoadReadings(
                initialTimeInterval, meterID);
1840          timer1SingleMeter.Interval = 15000;
1841          isFirstTimeSingleMeter = false;
1842          timer1SingleMeter.Start();
1843          //

1844          GetPrevioiusPeakForMeter(meterID);
1845          //

1846          return currentEnergyReadings;
1847      }


1848      /// <summary>
1849      /// Method to get previous peak based on meter
1850      /// </summary>
1851      /// <param name="meterID"></param>
1852      private void GetPrevioiusPeakForMeter(int meterID)
1853      {
1854          DateTime dateTimeFromThisDat = DateTime.Now.Subtract(new TimeSpan(0, 31,
                ));
1855          DateTime dateTimeToThisDat = DateTime.Now;
1856          int hours = Convert.ToInt32(spnEdtHours.EditValue);
1857          int minutes = Convert.ToInt32(spnEdtMinutes.EditValue);
1858          List<EnergyWindow> windows = energyWindowsRepository.
                LoadEnergyWindowForSingleMeter(eSelectionFormat.HoursMinutes,
                dateTimeFromThisDat, dateTimeToThisDat, hours, minutes, meterID);

1859          if (windows.Count > 0)
1860          {
1861              PreviousPeak = (float)(windows[0].energy_consumed);
1862              isPreviousPeackAvailable = true;
1863          }
1864          else
1865          {
1866              isPreviousPeackAvailable = false;
1867              MessageBox.Show("The current Cycle will not be forecast as previous pea
                    was not calculated!", "Wits Dasboard",
1868              MessageBoxButtons.OK, MessageBoxIcon.Warning);
1869          }
1870      }


1871      private List<DbEnergyReading> RefreshMultiMeterEnergyReadings(int
            initialTimeInterval, List<DbEnergyReading> currentEnergyReadings)
1872      {
1873          timerMultiMeterScreens.Stop();
1874          currentEnergyReadings = energyReadingRepository.LoadReadings(
                initialTimeInterval);
1875          timerMultiMeterScreens.Interval = 15000;
1876          timerMultiMeterScreens.Start();
1877          return currentEnergyReadings;
1878      }
```

```
1879      private List<DbPowerReading> RefreshSingleMeterPowerReadings(int
            initialTimeInterval, int meterID, List<DbPowerReading> currentReadings)
1880      {
1881        timer1SingleMeter.Stop();
1882        currentReadings = readingRepository.LoadReadings(initialTimeInterval,
              meterID);
1883        timer1SingleMeter.Interval = 15000;
1884        isFirstTimeSingleMeter = false;
1885        timer1SingleMeter.Start();
1886        return currentReadings;
1887      }

1888      private List<DbPowerReading> RefreshMultimeterPowerReadings(int
            initialTimeInterval, List<DbPowerReading> currentReadings)
1889      {
1890        timerMultiMeterScreens.Stop();
1891        currentReadings = readingRepository.LoadReadings(initialTimeInterval, -1);
1892        timerMultiMeterScreens.Interval = 15000;
1893        timerMultiMeterScreens.Start();
1894        return currentReadings;
1895      }

1896      private void repositoryItemGridLookUpEditAllMetersItem_EditValueChanged(
            object sender, EventArgs e)
1897      {
1898        btnSingleMeterStartLiveReadings.Enabled = true;
1899        btnSingleMeterStopLiveMetterReadings.Enabled = true;
1900        if (dateTimeSingleTimeFromThisDate.EditValue != null &&
              dateTimeSingleMeterToThisDate.EditValue != null && (
              repositoryItemGridLookUpEditAllMetersItem.EditValue != null))
1901        {
1902          btnLaunchSingleMeterArchive.Enabled = true;
1903          btnLaunchSingleMeterEnergRawEnergyReadings.Enabled = true;
1904        }
1905        if ((repositoryItemGridLookUpEditAllMetersItem.EditValue != null))
1906        {
1907          txtEditActiveMeter.EditValue = ((Meter)
                repositoryItemGridLookUpEditAllMetersItem.EditValue).Location;//.name;
1908        }
1909      }

1910      public void LoadMeterArchiveCharts()
1911      {
1912        // pivotGridControl1.DataSource = readingRepository.LoadMonthlyReadings(0,
              1, 1, 1);
1913        if (meterRepository.allMeterDetails != null)
1914        {
1915          readingRepository.MeterDetails = meterRepository.allMeterDetails;
1916        }
1917        List<PowerReading> dings = readingRepository.LoadMonthlyReadings(2, 1, 1,
            1);
1918        pivotGridControlSingleMeter.DataSource = dings;
1919        chartControlMeterPowerArchive.DataSource = pivotGridControlSingleMeter;
```

```
1920          //  Series seriesEnergy = chartControlMeterArchive.Series[0];
1921          // SeriesPoint[] pointsToUpdate = new SeriesPoint[dings.Count];
1922          /*for (int i = 0; i < dings.Count; i++)
1923          {
1924          PowerReading r = dings[i];
1925          pointsToUpdate[i] = new SeriesPoint(r.metertimestamp, r.powertotal);

1926        }-*/
1927          // seriesEnergy.Points.AddRange(pointsToUpdate);

1928        }
```

```
1929        /// <summary>
1930        /// Handler for loading single meter archive data
1931        /// </summary>
1932        /// <param name="sender"></param>
1933        /// <param name="e"></param>
1934        private void btnLaunchSingleMeterPowerArchive_ItemClick(object sender,
              ItemClickEventArgs e)
1935        {
1936          Cursor.Current = Cursors.WaitCursor;
1937          if (meterRepository.allMeterDetails != null)
1938          {
1939            readingRepository.MeterDetails = meterRepository.allMeterDetails;


1940            DateTime dateTimeFromThisDat = (DateTime)dateTimeSingleTimeFromThisDate
                  EditValue;
1941            DateTime dateTimeToThisDat = (DateTime)dateTimeSingleMeterToThisDate.
                  EditValue;
1942            List<PowerReading> dings = readingRepository.LoadPreviouseReadings(
                  dateTimeFromThisDat, dateTimeToThisDat, 1);
1943            pivotGridControlSingleMeter.DataSource = dings;
1944            chartControlMeterPowerArchive.DataSource = pivotGridControlSingleMeter;
1945            gridControlPowerReadings.DataSource = dings;
1946            dockPanelSingleMeterPowerArchive.DockAsMdiDocument();

1947          }
1948          Cursor.Current = Cursors.Default;
1949        }
```

```
1950        /// <summary>
1951        /// handler for date changed event
1952        /// </summary>
1953        /// <param name="sender"></param>
1954        /// <param name="e"></param>
1955        private void dateTimeToThisDate_EditValueChanged(object sender, EventArgs e)
1956        {
1957          if (dateTimeSingleTimeFromThisDate.EditValue != null &&
                dateTimeSingleMeterToThisDate.EditValue != null && (
                repositoryItemGridLookUpEditAllMetersItem.EditValue != null))
1958          {
1959            btnLaunchSingleMeterArchive.Enabled = true;
1960            btnLaunchSingleMeterEnergRawEnergyReadings.Enabled = true;
```

```
1961        }
1962    }

1963    /// <summary>
1964    /// Handler for launching all meter historical data
1965    /// </summary>
1966    /// <param name="sender"></param>
1967    /// <param name="e"></param>
1968    private void btnLaunchMultiMeterArchive_ItemClick(object sender,
          ItemClickEventArgs e)
1969    {
1970        if (meterRepository.allMeterDetails != null)
1971        {

1972            energyWindowsRepository.meters = meterRepository.allMeterDetails;


1973            DateTime dateTimeFromThisDat = (DateTime)
                  dateTimeMultiMeterTimeFromThisDate.EditValue;
1974            DateTime dateTimeToThisDat = (DateTime)dateTimeMultiMeterTimeToThisDate
                  EditValue;
1975            int hours = Convert.ToInt32(spnEdtHours.EditValue);
1976            int minutes = Convert.ToInt32(spnEdtMinutes.EditValue);
1977            List<EnergyWindow> windows = energyWindowsRepository.
                  LoadEnergyWindowReadings(eSelectionFormat.DateTime,
                  dateTimeFromThisDat, dateTimeToThisDat, hours, minutes);
1978            // pivotGridControlSingleMeter.DataSource = windows;
1979            pivotGridControlAllMetersArchive.DataSource = windows;
1980            // chartControlMeterArchive.DataSource = pivotGridControlSingleMeter;
1981            chartControlAllMetersEnergyWindowArchive.DataSource =
                  pivotGridControlAllMetersArchive;
1982            //  dockPanelAllMetersEnergyArchive.Show();
1983            if (!dockPanelAllMetersEnergyWindowsArchive.IsMdiDocument)
1984            {
1985                dockPanelAllMetersEnergyWindowsArchive.DockAsMdiDocument();
1986            }
1987        }
1988    }

1989    private void dateTimeMultiMeterTimeFromThisDate_EditValueChanged(object
          sender, EventArgs e)
1990    {
1991        if (dateTimeMultiMeterTimeFromThisDate.EditValue != null &&
              dateTimeMultiMeterTimeToThisDate != null)
1992        {
1993            btnLaunchMultiMeterEnergyWindowsArchive.Enabled = true;
1994            btnLaunchMultiMeterEnergyReadingsArchive.Enabled = true;
1995        }
1996    }

1997    private void dateTimeMultiMeterTimeToThisDate_EditValueChanged(object sende
          , EventArgs e)
1998    {
```

```
1999        if (dateTimeMultiMeterTimeFromThisDate.EditValue != null &&
              dateTimeMultiMeterTimeToThisDate != null)
2000        {
2001          btnLaunchMultiMeterEnergyWindowsArchive.Enabled = true;
2002        }
2003      }

2004      /// <summary>
2005      /// Method to set up multi meter archive UI
2006      /// </summary>
2007      private void SetUpMultiMeterArchiveUI()
2008      {
2009        chkEditUseDateSelection.EditValue = false;
2010        chkEditUseTimeSelection.EditValue = true;
2011        dateTimeMultiMeterTimeToThisDate.EditValue = DateTime.Today;
2012        dateTimeMultiMeterTimeFromThisDate.EditValue = DateTime.Today;
2013      }

2014      private void chkEditUseDateSelection_EditValueChanged(object sender,
            EventArgs e)
2015      {
2016        ToggleSelectionOptionsForMultiMeter();
2017      }

2018      private void ToggleSelectionOptionsForMultiMeter()
2019      {
2020        this.chkEditUseDateSelection.EditValueChanged -= new System.EventHandler(
              this.chkEditUseDateSelection_EditValueChanged);
2021        this.chkEditUseTimeSelection.EditValueChanged -= new System.EventHandler(
              this.chkEditUseTimeSelection_EditValueChanged);
2022        chkEditUseDateSelection.EditValue = !(bool)chkEditUseTimeSelection.
              EditValue;
2023        chkEditUseTimeSelection.EditValue = !(bool)chkEditUseDateSelection.
              EditValue;
2024        this.chkEditUseDateSelection.EditValueChanged += new System.EventHandler(
              this.chkEditUseDateSelection_EditValueChanged);
2025        this.chkEditUseTimeSelection.EditValueChanged += new System.EventHandler(
              this.chkEditUseTimeSelection_EditValueChanged);
2026      }

2027      private void chkEditUseTimeSelection_EditValueChanged(object sender,
            EventArgs e)
2028      {
2029        ToggleSelectionOptionsForMultiMeter();
2030      }

2031      /// <summary>
2032      /// Handler to load the grid with energy readings.
2033      /// </summary>
2034      /// <param name="sender"></param>
2035      /// <param name="e"></param>
```

```
2036      private void btnLoadSingleMeterArchivedEnergyReadings_ItemClick(object
            sender, ItemClickEventArgs e)
2037      {
2038        Cursor.Current = Cursors.WaitCursor;
2039        if (meterRepository.allMeterDetails != null)
2040        {
2041          //
2042          pivotGridControlSingleMeterEnergyArchive.DataSource = null;
2043          chartControlMeterEnergyArchive.DataSource = null;
2044          gridControlRawEnergyReadings.DataSource = null;
2045          pivotGridControlSingleMeterEnergyArchive.CollapseAll();
2046          //
2047          Meter selectedMeter = null;
2048          energyReadingRepository.MeterDetails = meterRepository.allMeterDetails;
2049          if (dateTimeSingleTimeFromThisDate.EditValue != null &&
                dateTimeSingleMeterToThisDate.EditValue != null && (
                repositoryItemGridLookUpEditAllMetersItem.EditValue != null))
2050          {
2051            selectedMeter = repositoryItemGridLookUpEditAllMetersItem.EditValue a
                 Meter;
2052          }
2053          if (selectedMeter != null)
2054          {
2055            DateTime dateTimeFromThisDat = (DateTime)
                  dateTimeSingleTimeFromThisDate.EditValue;
2056            DateTime dateTimeToThisDat = (DateTime)dateTimeSingleMeterToThisDate.
                  EditValue;
2057            List<EnergyReading> dings = energyReadingRepository.
                  LoadEnergyReadingsBasedOnDates(dateTimeFromThisDat,
                  dateTimeToThisDat, (double)spnEditorEnergyPricePerKwhr.EditValue,
                  selectedMeter.id);

2058            if (dings.Count > 0)
2059            {

2060              pivotGridControlSingleMeterEnergyArchive.DataSource = dings;
2061              chartControlMeterEnergyArchive.DataSource =
                    pivotGridControlSingleMeterEnergyArchive;
2062              gridControlRawEnergyReadings.DataSource = dings;
2063              documentManager.BeginUpdate();
2064              if (!dockPanelRawEnergyReadings.IsMdiDocument)
2065              {
2066                dockPanelRawEnergyReadings.DockAsMdiDocument();
2067              }
2068              if (!dockPanelMeterEnergyReadingsArchive.IsMdiDocument)
2069              {
2070                dockPanelMeterEnergyReadingsArchive.DockAsMdiDocument();
2071              }
2072              documentManager.EndUpdate();
2073            }
2074            else
2075            {

2076              MessageBox.Show("The selected meter does not have any readings in
                    the selected range!", "Wits Dasboard",
2077              MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
2078            }
2079          }
2080        }

2081        Cursor.Current = Cursors.Default;
2082      }


2083      /// <summary>
2084      /// Handler to launch multi meter split screen Live readings.
2085      /// </summary>
2086      /// <param name="sender"></param>
2087      /// <param name="e"></param>
2088      private void btnMultiMeterLiveReadingsLaunch_ItemClick(object sender,
            ItemClickEventArgs e)
2089      {
2090        timerMultiMeterScreens.Enabled = false;
2091        timerMultiMeterScreens.Stop();
2092        SetUpRepositorys();
2093        LoadMeters();
2094        repositoryItemGridLookUpEditShowHideMeterCharts.DataSource = Meters;
2095        //  isFirstTimeMultiMeter = true;
2096        // Series series = chartControlPowerReadings.Series[0];
2097        //Series energySeries = chartControlEnergyReadings.Series[0];
2098        // series.Points.Clear();
2099        //energySeries.Points.Clear();
2100        //
2101        chartControlSplit1.Series[0].Points.Clear();
2102        chartControlSplit2.Series[0].Points.Clear();
2103        chartControlSplit3.Series[0].Points.Clear();
2104        chartControlSplit4.Series[0].Points.Clear();
2105        chartControlSplit5.Series[0].Points.Clear();
2106        chartControlSplit6.Series[0].Points.Clear();
2107        chartControlSplit7.Series[0].Points.Clear();
2108        chartControlSplit8.Series[0].Points.Clear();

2109        //
2110        initialTimeInterval = 60;
2111        initialEnergyTimeInterval = 600;
2112        //meterID = m.id;
2113        shortTimeInterval = 5;
2114        LoadAllMultiMeterReadings(initialTimeInterval, initialEnergyTimeInterval,
              shortTimeInterval);

2115        timerMultiMeterScreens.Enabled = true;
2116        timer1SingleMeter.Enabled = false;
2117        timerMultiMeterScreens.Start();
2118        isFirstTimeMultiMeter = false;

2119        if (!dockPanelMultiMeterSplitLiveGraphs.IsMdiDocument)
2120        {
2121          dockPanelMultiMeterSplitLiveGraphs.DockAsMdiDocument();
2122        }
2123      }
```

```
2124      private void timer1SingleMeter_Tick(object sender, EventArgs e)
2125      {
2126        timerMultiMeterScreens.Stop();//avoid both timers running same time
2127        timer1SingleMeter.Stop();
2128        Cursor.Current = Cursors.WaitCursor;
2129        LoadAllSingleMeterReadings(initialTimeInterval, initialEnergyTimeInterval
              meterID, shortTimeInterval);
2130        timer1SingleMeter.Start();
2131        SwiftPlotDiagram diagram = (SwiftPlotDiagram)chartControlEnergyReadings.
            Diagram;
2132        diagram.AxisY.ConstantLines[0].AxisValue = spnEditPeakEnergyLine.EditValu
            ;
2133        SwiftPlotDiagram diagram1 = (SwiftPlotDiagram)chartControlPowerReadings.
            Diagram;
2134        diagram1.AxisY.ConstantLines[0].AxisValue = spnEditPeakPowerValue.
            EditValue;
2135        Cursor.Current = Cursors.Default;
2136      }

2137      private void timerMultiMeterScreens_Tick(object sender, EventArgs e)
2138      {
2139        timer1SingleMeter.Stop();//avoid two timers running at the same time
2140        timerMultiMeterScreens.Stop();//lets not count loading time
2141        Cursor.Current = Cursors.WaitCursor;
2142        LoadAllMultiMeterReadings(initialTimeInterval, initialEnergyTimeInterval,
            shortTimeInterval);
2143        Cursor.Current = Cursors.Default;
2144      }

2145      private void btnSingleMeterStopLiveMetterReadings_ItemClick(object sender,
            ItemClickEventArgs e)
2146      {
2147        //chartControlSplit1.Series[0].ChangeView(ViewType.SwiftPlot);

2148        if (timer1SingleMeter.Enabled)
2149        {
2150          timer1SingleMeter.Stop();
2151          btnSingleMeterStopLiveMetterReadings.Caption = "Resume Live Readings";
2152        }
2153        else
2154        {
2155          timer1SingleMeter.Enabled = true;
2156          timer1SingleMeter.Start();
2157          btnSingleMeterStopLiveMetterReadings.Caption = "Pause Live Readings";
2158        }
2159      }

2160      private void btnMultiMeterLiveReadingsStop_ItemClick(object sender,
            ItemClickEventArgs e)
2161      {
2162        if (timerMultiMeterScreens.Enabled)
2163        {
2164          timerMultiMeterScreens.Enabled = false;
```

```
2165            timerMultiMeterScreens.Stop();
2166            btnMultiMeterLiveReadingsStop.Caption = "Resume Live Readings";
2167          }
2168          else
2169          {
2170            timerMultiMeterScreens.Enabled = true;
2171            timerMultiMeterScreens.Start();
2172            btnMultiMeterLiveReadingsStop.Caption = "Pause Live Readings";
2173          }
2174        }

2175        private void gridLookUpEditShowHideMeterCharts_EditValueChanged(object
              sender, EventArgs e)
2176        {
2177          BarEditItem edit = sender as BarEditItem;
2178          if (edit.EditValue.GetType() == typeof(Meter))
2179          {
2180            ToggleMeterChatDisplay(edit.EditValue as Meter);
2181          }
2182        }

2183        private void ToggleMeterChatDisplay(Meter meter)
2184        {
2185          switch (meter.id)
2186          {
2187            case 1:

2188              if (meter.IsShown)
2189              {
2190                layoutControlItemchartControlSplit1.Visibility = LayoutVisibility.
                    Never;

2191                splitterItem1And8.Visibility = LayoutVisibility.Never;
2192                meter.IsShown = false;
2193              }
2194              else
2195              {
2196                layoutControlItemchartControlSplit1.Visibility = LayoutVisibility.
                      Always;
2197                if (layoutControlItemchartControlSplit8.Visible ||
                      layoutControlItemchartControlSplit4.Visible ||
                      layoutControlItemchartControlSplit3.Visible)
2198                {
2199                  splitterItem1And8.Visibility = LayoutVisibility.Always;
2200                }
2201                meter.IsShown = true;
2202              }

2203              break;

2204            case 2:
2205              if (meter.IsShown)
2206              {
2207                layoutControlItemchartControlSplit2.Visibility = LayoutVisibility.
```

```
                Never;
2208            splitterItem7and2.Visibility = LayoutVisibility.Never;
2209            meter.IsShown = false;
2210          }
2211          else
2212          {
2213            layoutControlItemchartControlSplit2.Visibility = LayoutVisibility.
                  Always;
2214            if (layoutControlItemchartControlSplit7.Visible ||
                  layoutControlItemchartControlSplit5.Visible ||
                  layoutControlItemchartControlSplit6.Visible)
2215            {
2216              splitterItem7and2.Visibility = LayoutVisibility.Always;
2217            }
2218            if (layoutControlItemchartControlSplit6.Visible)
2219            {
2220              splitterItem6and5.Visibility = LayoutVisibility.Always;

2221            }
2222            meter.IsShown = true;
2223          }
2224          break;
2225        case 3:
2226          if (meter.IsShown)
2227          {
2228            layoutControlItemchartControlSplit3.Visibility = LayoutVisibility.
                  Never;
2229            splitterItem4and3.Visibility = LayoutVisibility.Never;
2230            meter.IsShown = false;

2231          }
2232          else
2233          {
2234            layoutControlItemchartControlSplit3.Visibility = LayoutVisibility.
                  Always;
2235            if (layoutControlItemchartControlSplit8.Visible ||
                  layoutControlItemchartControlSplit4.Visible ||
                  layoutControlItemchartControlSplit1.Visible)
2236            {
2237              splitterItem4and3.Visibility = LayoutVisibility.Always;
2238            }

2239            if (layoutControlItemchartControlSplit1.Visible)
2240            {
2241              splitterItem1And8.Visibility = LayoutVisibility.Always;
2242            }
2243            meter.IsShown = true;
2244          }
2245          break;
2246        case 4:
2247          if (meter.IsShown)
2248          {
2249            layoutControlItemchartControlSplit4.Visibility = LayoutVisibility.
                  Never;
2250            if (!layoutControlItemchartControlSplit3.Visible)
2251            {
```

```
2252                splitterItem4and3.Visibility = LayoutVisibility.Never;
2253            }
2254            if (!layoutControlItemchartControlSplit8.Visible)
2255            {
2256                splitterItem8and4.Visibility = LayoutVisibility.Never;
2257            }
2258            meter.IsShown = false;
2259        }
2260        else
2261        {

2262            layoutControlItemchartControlSplit4.Visibility = LayoutVisibility.
                  Always;
2263            if (layoutControlItemchartControlSplit3.Visible)
2264            {
2265                splitterItem4and3.Visibility = LayoutVisibility.Always;
2266            }
2267            if (layoutControlItemchartControlSplit8.Visible)
2268            {
2269                splitterItem8and4.Visibility = LayoutVisibility.Always;
2270            }
2271            if (layoutControlItemchartControlSplit1.Visible)
2272            {
2273                splitterItem1And8.Visibility = LayoutVisibility.Always;
2274            }
2275            meter.IsShown = true;
2276        }
2277        break;
2278    case 5:
2279        if (meter.IsShown)
2280        {
2281            layoutControlItemchartControlSplit5.Visibility = LayoutVisibility.
                  Never;
2282            if (!layoutControlItemchartControlSplit7.Visible)
2283            {
2284                splitterItem5and7.Visibility = LayoutVisibility.Never;
2285            }
2286            if (!layoutControlItemchartControlSplit6.Visible)
2287            {
2288                splitterItem6and5.Visibility = LayoutVisibility.Never;
2289            }
2290            meter.IsShown = false;
2291        }
2292        else
2293        {
2294            layoutControlItemchartControlSplit5.Visibility = LayoutVisibility.
                  Always;
2295            if (layoutControlItemchartControlSplit7.Visible)
2296            {
2297                splitterItem5and7.Visibility = LayoutVisibility.Always;
2298            }
2299            if (layoutControlItemchartControlSplit6.Visible)
2300            {
2301                splitterItem6and5.Visibility = LayoutVisibility.Always;
2302            }
2303            meter.IsShown = true;
```

```
2304            }
2305            break;
2306        case 6:
2307            if (meter.IsShown)
2308            {
2309                layoutControlItemchartControlSplit6.Visibility = LayoutVisibility.
                    Never;
2310                splitterItem6and5.Visibility = LayoutVisibility.Never;
2311                meter.IsShown = false;
2312            }
2313            else
2314            {
2315                layoutControlItemchartControlSplit6.Visibility = LayoutVisibility.
                    Always;
2316                if (layoutControlItemchartControlSplit5.Visible ||
                    layoutControlItemchartControlSplit7.Visible ||
                    layoutControlItemchartControlSplit2.Visible)
2317                {
2318                    splitterItem6and5.Visibility = LayoutVisibility.Always;
2319                }
2320                meter.IsShown = true;
2321            }
2322            break;
2323        case 7:
2324            if (meter.IsShown)
2325            {
2326                layoutControlItemchartControlSplit7.Visibility = LayoutVisibility.
                    Never;
2327                if (!layoutControlItemchartControlSplit5.Visible)
2328                {
2329                    splitterItem5and7.Visibility = LayoutVisibility.Never;
2330                }
2331                if (!layoutControlItemchartControlSplit2.Visible)
2332                {
2333                    splitterItem7and2.Visibility = LayoutVisibility.Never;
2334                }
2335                meter.IsShown = false;
2336            }
2337            else
2338            {
2339                layoutControlItemchartControlSplit7.Visibility = LayoutVisibility.
                    Always;
2340                if (layoutControlItemchartControlSplit2.Visible)
2341                {
2342                    splitterItem7and2.Visibility = LayoutVisibility.Always;
2343                }
2344                if (layoutControlItemchartControlSplit5.Visible)
2345                {
2346                    splitterItem5and7.Visibility = LayoutVisibility.Always;
2347                }
2348                if (layoutControlItemchartControlSplit6.Visible)
2349                {
2350                    splitterItem6and5.Visibility = LayoutVisibility.Always;

2351                }
2352                meter.IsShown = true;
```

```
2353            }
2354            break;
2355        case 8:
2356            if (meter.IsShown)
2357            {
2358                layoutControlItemchartControlSplit8.Visibility = LayoutVisibility.
                    Never;
2359                if (!layoutControlItemchartControlSplit1.Visible)
2360                {
2361                    splitterItem1And8.Visibility = LayoutVisibility.Never;
2362                }
2363                if (!layoutControlItemchartControlSplit4.Visible)
2364                {
2365                    splitterItem8and4.Visibility = LayoutVisibility.Never;
2366                }
2367                meter.IsShown = false;
2368            }
2369            else
2370            {
2371                layoutControlItemchartControlSplit8.Visibility = LayoutVisibility.
                    Always;
2372                if (layoutControlItemchartControlSplit1.Visible)
2373                {
2374                    splitterItem1And8.Visibility = LayoutVisibility.Always;
2375                }
2376                if (layoutControlItemchartControlSplit4.Visible)
2377                {
2378                    splitterItem8and4.Visibility = LayoutVisibility.Always;
2379                }
2380                meter.IsShown = true;
2381            }
2382            break;
2383        default:
2384            break;
2385    }
2386}
```

```
2387    private void btnSingleMeterStartLiveReadings_ItemClick(object sender,
          ItemClickEventArgs e)
2388    {
2389        Cursor.Current = Cursors.WaitCursor;
2390        SetUpRepositorys();
2391        LoadMeters();
2392        SetConstantEnergyConstantLineForChart(chartControlEnergyReadings);
2393        SePeakPowerConstantLineForLivePowerChart(chartControlPowerReadings);

2394        Meter m = repositoryItemGridLookUpEditAllMetersItem.EditValue as Meter;
2395        if (m != null)
2396        {
2397            isFirstTimeSingleMeter = true;
2398            gmap.Position = Location(m.id);
2399            gmap.Zoom = 19;
2400            Series series = chartControlPowerReadings.Series[0];
2401            Series energySeries = chartControlEnergyReadings.Series[0];
2402            series.Points.Clear();//.RemoveRange(0, series.Points.Count);
```

```
2403            energySeries.Points.Clear();
2404            initialTimeInterval = 60;
2405            initialEnergyTimeInterval = 600;
2406            meterID = m.id;
2407            shortTimeInterval = 5;
2408            LoadAllSingleMeterReadings(initialTimeInterval,
                  initialEnergyTimeInterval, meterID, shortTimeInterval);
2409        }
2410        timer1SingleMeter.Enabled = true;
2411        timerMultiMeterScreens.Enabled = false;//just making sure we dont have to
              many timer running
2412        dockPanelSingleMeterLiveEnergyReadings.DockAsMdiDocument();
2413        dockPanelSingleMeterLivePoower.DockAsMdiDocument();
2414        Cursor.Current = Cursors.Default;
2415    }
```

```
2416    /// <summary>
2417    /// Method to set the constan line for the energy chart.
2418    /// constant line is the peak value line.
2419    /// </summary>
2420    /// <param name="chartControlEnergyReadings"></param>
2421    private void SetConstantEnergyConstantLineForChart(ChartControl
          chartControlEnergyReadings)
2422    {
2423        SwiftPlotDiagram diagram = (SwiftPlotDiagram)chartControlEnergyReadings.
              Diagram;
2424        diagram.AxisY.ConstantLines[0].AxisValue = spnEditPeakEnergyLine.EditValu
              ;
2425    }
```

```
2426    /// <summary>
2427    /// method to set peak power constant line for live power chart
2428    /// </summary>
2429    /// <param name="chartControlPowerReadings"></param>
2430    private void SePeakPowerConstantLineForLivePowerChart(ChartControl
          chartControlPowerReadings)
2431    {
2432        SwiftPlotDiagram diagram = (SwiftPlotDiagram)chartControlPowerReadings.
              Diagram;
2433        diagram.AxisY.ConstantLines[0].AxisValue = spnEditPeakPowerValue.EditValu
              ;
2434    }
```

```
2435    /// <summary>
2436    /// Method to set up the available chart types
2437    /// </summary>
2438    private void SetUpChartTypes()
2439    {
2440        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Bar);
2441        repositoryItemComboBoxChartTypes.Items.Add(ViewType.SideBySideStackedBar3
              );
2442        repositoryItemComboBoxChartTypes.Items.Add(ViewType.FullStackedBar);
2443        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Pie3D);
```

```
2444        repositoryItemComboBoxChartTypes.Items.Add(ViewType.FullStackedLine3D);
2445        repositoryItemComboBoxChartTypes.Items.Add(ViewType.FullStackedLine);
2446        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Pie);
2447        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Area3D);
2448        repositoryItemComboBoxChartTypes.Items.Add(ViewType.ManhattanBar);
2449        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Area3D);
2450        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Bar3D);
2451        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Funnel3D);
2452        repositoryItemComboBoxChartTypes.Items.Add(ViewType.FullStackedArea3D);
2453        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Line3D);
2454        repositoryItemComboBoxChartTypes.Items.Add(ViewType.StackedArea3D);
2455        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Bubble);
2456        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Doughnut);
2457        repositoryItemComboBoxChartTypes.Items.Add(ViewType.SideBySideStackedBar3
              );
2458        repositoryItemComboBoxChartTypes.Items.Add(ViewType.StackedLine);
2459        repositoryItemComboBoxChartTypes.Items.Add(ViewType.SideBySideStackedBar);
2460        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Spline3D);
2461        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Spline);
2462        repositoryItemComboBoxChartTypes.Items.Add(ViewType.SplineArea3D);
2463        repositoryItemComboBoxChartTypes.Items.Add(ViewType.StackedSplineArea);
2464        repositoryItemComboBoxChartTypes.Items.Add(ViewType.StackedSplineArea3D);
2465        repositoryItemComboBoxChartTypes.Items.Add(ViewType.SwiftPlot);
2466        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Line);
2467        repositoryItemComboBoxChartTypes.Items.Add(ViewType.Point);

2468        comboBoxEdtChartTypes.EditValue = ViewType.Bar3D;
2469    }
```

```
2470    private void comboBoxEdtChartTypes_EditValueChanged(object sender, EventArg
            e)
2471    {
2472        // comboBoxEdtChartTypes.EditValue
2473        //      chartControlSplit1.Series[0].ChangeView(ViewType.SwiftPlot);
2474        //
    ucPivotGridChartControl.ChartControl.SeriesTemplate.ChangeView((ViewType)((itmcbChartView


2475        chartControlMeterEnergyArchive.SeriesTemplate.ChangeView((ViewType)((
              comboBoxEdtChartTypes.EditValue)));
2476        chartControlMeterPowerArchive.SeriesTemplate.ChangeView((ViewType)((
              comboBoxEdtChartTypes.EditValue)));
2477        chartControlAllMetersEnergyWindowArchive.SeriesTemplate.ChangeView((
              ViewType)((comboBoxEdtChartTypes.EditValue)));
2478        Diagram3D diagram1 = chartControlMeterEnergyArchive.Diagram as Diagram3D;
2479        if (diagram1 != null)
2480        {
2481            diagram1.RuntimeRotation = true;
2482            diagram1.RuntimeScrolling = true;
2483            diagram1.RuntimeZooming = true;

2484        }
2485        else
2486        {
2487            XYDiagram diagram11 = chartControlMeterEnergyArchive.Diagram as
```

```
            XYDiagram;
2488        if (diagram11 != null)
2489        {
2490          diagram11.EnableAxisXScrolling = true;
2491          diagram11.EnableAxisXZooming = true;
2492          diagram11.EnableAxisYScrolling = true;
2493          diagram11.EnableAxisYZooming = true;
2494        }
2495        else
2496        {
2497          XYDiagram3D diagram3d = chartControlMeterEnergyArchive.Diagram as
              XYDiagram3D;
2498          if (diagram3d != null)
2499          {
2500            diagram3d.RuntimeScrolling = true;
2501            diagram3d.RuntimeZooming = true;
2502            diagram3d.RuntimeRotation = true;
2503          }
2504          else
2505          {

2506            SwiftPlotDiagram diagramsp = chartControlMeterEnergyArchive.Diagram
              as SwiftPlotDiagram;
2507            if (diagramsp != null)
2508            {
2509              diagramsp.EnableAxisXScrolling = true;
2510              //  diagramsp.EnableAxisYScrolling = true;
2511              diagramsp.EnableAxisXZooming = true;

2512              diagramsp.AxisX.NumericOptions.Format = DevExpress.XtraCharts.
                NumericFormat.General;
2513              diagramsp.AxisX.Range.ScrollingRange.SideMarginsEnabled = true;
2514              diagramsp.AxisX.Range.SideMarginsEnabled = true;
2515              diagramsp.AxisX.DateTimeGridAlignment = DevExpress.XtraCharts.
                DateTimeMeasurementUnit.Minute;
2516              diagramsp.AxisX.DateTimeMeasureUnit = DevExpress.XtraCharts.
                DateTimeMeasurementUnit.Minute;
2517              diagramsp.AxisX.DateTimeOptions.Format = DevExpress.XtraCharts.
                DateTimeFormat.Custom;
2518              diagramsp.AxisX.DateTimeOptions.FormatString = "MMM d HH:mm:ss";
2519              diagramsp.AxisX.GridLines.Visible = true;
2520              diagramsp.AxisX.NumericOptions.Format = DevExpress.XtraCharts.
                NumericFormat.General;
2521              diagramsp.AxisX.Range.ScrollingRange.SideMarginsEnabled = true;
2522              diagramsp.AxisX.Range.SideMarginsEnabled = true;
2523              diagramsp.AxisX.Tickmarks.CrossAxis = true;
2524              diagramsp.AxisX.VisibleInPanesSerializable = "-1";

2525            }

2526          }
2527        }
2528      }


2529      Diagram3D diagram2 = chartControlMeterPowerArchive.Diagram as Diagram3D;
```

```
2530        if (diagram2 != null)
2531        {
2532          diagram2.RuntimeRotation = true;
2533          diagram2.RuntimeScrolling = true;
2534          diagram2.RuntimeZooming = true;

2535        }
2536        else
2537        {
2538          XYDiagram diagram211 = chartControlMeterPowerArchive.Diagram as
                XYDiagram;
2539          if (diagram211 != null)
2540          {
2541            diagram211.EnableAxisXScrolling = true;
2542            diagram211.EnableAxisXZooming = true;
2543            diagram211.EnableAxisYScrolling = true;
2544            diagram211.EnableAxisYZooming = true;
2545          }

2546          else
2547          {
2548            XYDiagram3D diagram3d = chartControlMeterEnergyArchive.Diagram as
                  XYDiagram3D;
2549            if (diagram3d != null)
2550            {
2551              diagram3d.RuntimeScrolling = true;
2552              diagram3d.RuntimeZooming = true;
2553              diagram3d.RuntimeRotation = true;

2554            }
2555          }
2556        }
2557        Diagram3D diagram3 = chartControlAllMetersEnergyWindowArchive.Diagram as
              Diagram3D;
2558        if (diagram3 != null)
2559        {
2560          diagram3.RuntimeRotation = true;
2561          diagram3.RuntimeScrolling = true;
2562          diagram3.RuntimeZooming = true;

2563        }
2564        else
2565        {
2566          XYDiagram diagram311 = chartControlAllMetersEnergyWindowArchive.Diagram
                as XYDiagram;
2567          if (diagram311 != null)
2568          {
2569            diagram311.EnableAxisXScrolling = true;
2570            diagram311.EnableAxisXZooming = true;
2571            diagram311.EnableAxisYScrolling = true;
2572            diagram311.EnableAxisYZooming = true;
2573          }
2574          else
2575          {
2576            XYDiagram3D diagram3d = chartControlMeterEnergyArchive.Diagram as
                  XYDiagram3D;
```

```
2577            if (diagram3d != null)
2578            {
2579              diagram3d.RuntimeScrolling = true;
2580              diagram3d.RuntimeZooming = true;
2581              diagram3d.RuntimeRotation = true;
2582            }
2583          }
2584        }
2585        ViewType vt = ((ViewType)((comboBoxEdtChartTypes.EditValue)));
2586        if (vt == ViewType.SwiftPlot)
2587        {
2588          List<EnergyReading> readings = pivotGridControlSingleMeterEnergyArchive
              DataSource as List<EnergyReading>;
2589          //

2590          if (readings != null)
2591          {
2592            Series seriesEnergy = chartControlMeterEnergyArchive.Series[0];
2593            SeriesPoint[] pointsToUpdate = new SeriesPoint[readings.Count];
2594            for (int i = 0; i < readings.Count; i++)
2595            {
2596              EnergyReading r = readings[i];
2597              pointsToUpdate[i] = new SeriesPoint(r.measurement_timestamp, r.valu
                  );


2598            }
2599            seriesEnergy.Points.AddRange(pointsToUpdate);
2600          }
2601          //



2602        }
2603      }
```

```
2604      /// <summary>
2605      /// Button to load archived enery readings for all the meters.
2606      /// </summary>
2607      /// <param name="sender"></param>
2608      /// <param name="e"></param>
2609      private void btnLaunchMultiMeterEnergyReadingsArchive_ItemClick(object
            sender, ItemClickEventArgs e)
2610      {
2611        Cursor.Current = Cursors.WaitCursor;
2612        if (meterRepository.allMeterDetails != null)
2613        {
2614          pivotGridControlSingleMeterEnergyArchive.DataSource = null;
2615          chartControlMeterEnergyArchive.DataSource = null;
2616          gridControlRawEnergyReadings.DataSource = null;
2617          //  pivotGridControlSingleMeterEnergyArchive.CollapseAll();
2618          pivotGridControlSingleMeterEnergyArchive.CollapseAll();
2619          //  pivotGridControlAllMetersArchive.CollapseAll();
2620          energyReadingRepository.MeterDetails = meterRepository.allMeterDetails;

2621          DateTime dateTimeFromThisDat = (DateTime)
              dateTimeMultiMeterTimeFromThisDate.EditValue;
```

```
2622            DateTime dateTimeToThisDat = (DateTime)dateTimeMultiMeterTimeToThisDate
                  EditValue;
2623            List<EnergyReading> dings = energyReadingRepository.
                  LoadEnergyReadingsBasedOnDates(dateTimeFromThisDat, dateTimeToThisDat
                   (double)spnEditorEnergyPricePerKwhr.EditValue);

2624            if (dings.Count > 0)
2625            {

2626              pivotGridControlSingleMeterEnergyArchive.DataSource = dings;
2627              chartControlMeterEnergyArchive.DataSource =
                    pivotGridControlSingleMeterEnergyArchive;
2628              gridControlRawEnergyReadings.DataSource = dings;
2629              documentManager.BeginUpdate();
2630              if (!dockPanelRawEnergyReadings.IsMdiDocument)
2631              {
2632                dockPanelRawEnergyReadings.DockAsMdiDocument();
2633              }
2634              if (!dockPanelMeterEnergyReadingsArchive.IsMdiDocument)
2635              {
2636                dockPanelMeterEnergyReadingsArchive.DockAsMdiDocument();
2637              }
2638              documentManager.EndUpdate();
2639            }
2640            else
2641            {

2642              MessageBox.Show("There are no archived readings in the selected time
                    range!", "Wits Dasboard",
2643              MessageBoxButtons.OK, MessageBoxIcon.Error);
2644            }

2645          }

2646            Cursor.Current = Cursors.Default;
2647        }
```

```
2648        /// <summary>
2649        /// Handler to launch the weather info browser
2650        /// </summary>
2651        /// <param name="sender"></param>
2652        /// <param name="e"></param>
2653        private void btnLlaunchWeatherCenter_ItemClick(object sender,
              ItemClickEventArgs e)
2654        {
2655          webBrowserWeather.Navigate("http://www.weathersa.co.za/web/index.php");
2656          if (!dockPanelWeatherCenter.IsMdiDocument)
2657          {
2658            dockPanelWeatherCenter.DockAsMdiDocument();
2659          }
2660        }
```

```
2661    }
2662  }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Data.Entity;
6  using Models.Enumerations;
7  using Models;

8  namespace DataAccess
9  {
10   public class EnergyWindowsRepository : DbContext
11   {
12     public Dictionary<int, Meter> meters = null;

13     public DbSet<DbEnergyWindow> EnergyWindows {
14     }

15     public EnergyWindowsRepository(Npgsql.NpgsqlConnection conn)
16     : base(conn, true)
17     {
18     }

19     protected override void OnModelCreating(DbModelBuilder modelBuilder)
20     {
21       modelBuilder.Entity<DbEnergyWindow>().ToTable("energy_windows", "public");
22     }


23     /// <summary>
24     /// Load EnergyWindows from the past x minutes
25     /// </summary>
26     /// <param name="elapsedTime"></param>
27     /// <returns></returns>
28     public List<EnergyWindow> LoadEnergyWindowReadings(eSelectionFormat
         selection, DateTime startTime,DateTime endTime,int elapsedhour,int
         elapsedMinutes)
29     {
30       List<DbEnergyWindow> dbwindows = new List<DbEnergyWindow>();
31       List<EnergyWindow> windows = new List<EnergyWindow>();

32       switch (selection)
33       {
34         case eSelectionFormat.DateTime:
35           GetWindowsForAllMetersBasedOnDate(startTime,endTime, dbwindows,
               windows);
36           break;
37         case eSelectionFormat.HoursMinutes:
38           GetWindowsForAllMetersBasedOnTime(elapsedhour, elapsedMinutes,
               dbwindows, windows);
39           break;
40         default:
41           break;
42       }


43       return windows;
```

```
44      }

45      public List<EnergyWindow> LoadEnergyWindowForSingleMeter(eSelectionFormat
          selection, DateTime startTime, DateTime endTime, int elapsedhour, int
          elapsedMinutes,int meterID)
46      {
47        List<DbEnergyWindow> dbwindows = new List<DbEnergyWindow>();
48        List<EnergyWindow> windows = new List<EnergyWindow>();

49        switch (selection)
50        {
51          case eSelectionFormat.DateTime:
52            GetWindowsForThisMeterBasedOnDate(startTime,meterID, dbwindows,
                windows);
53            break;
54          case eSelectionFormat.HoursMinutes:
55            GetWindowsForThisMeterBasedOnTime(elapsedhour, elapsedMinutes,meterID,
                dbwindows, windows);
56            break;
57          default:
58            break;
59        }


60        return windows;
61      }

62      private void GetWindowsForThisMeterBasedOnDate(DateTime startTime, int
          meterID, List<DbEnergyWindow> dbwindows, List<EnergyWindow> windows)
63      {
64        var selectedReadings = from read in this.EnergyWindows
65          where read.start_time > startTime && read.meter_id == meterID
66          select read;

67        int count = selectedReadings.Count();
68        foreach (var item in selectedReadings)
69        {
70          dbwindows.Add(item);
71        }
72        foreach (var item in dbwindows)
73        {
74          windows.Add(new EnergyWindow()
75          {
76            id = item.id,
77            Meter = meters[item.meter_id],
78            start_time = item.start_time,
79            end_time=item.end_time,
80            energy_consumed = item.energy_consumed,
81            //is_estimate = item.is_estimate,
82            is_fully_populated = item.is_fully_populated
83          });
84        }
85      }
```

```
86      private void GetWindowsForThisMeterBasedOnTime(int hour,int minute, int
          meterID, List<DbEnergyWindow> dbwindows, List<EnergyWindow> windows)
87      {
88        TimeSpan ts = new TimeSpan(0, 31, 0);

89        DateTime time = DateTime.Now.Subtract(ts);
90        var selectedReadings = from read in this.EnergyWindows
91          where read.end_time > time && read.meter_id == meterID
92          select read;

93        int count = selectedReadings.Count();
94        foreach (var item in selectedReadings)
95        {
96          dbwindows.Add(item);
97        }
98        foreach (var item in dbwindows)
99        {
100         windows.Add(new EnergyWindow()
101         {
102           id = item.id,
103           Meter = meters[item.meter_id],
104           start_time = item.start_time,
105           energy_consumed = item.energy_consumed,
106           // is_estimate = item.is_estimate,
107           is_fully_populated = item.is_fully_populated
108         });
109       }
110     }
```

```
111     private void GetWindowsForAllMetersBasedOnDate(DateTime startTime,DateTime
          endTime, List<DbEnergyWindow> dbwindows, List<EnergyWindow> windows)
112     {
113       var selectedReadings = from read in this.EnergyWindows
114         where  read.start_time > startTime && read.start_time< endTime
115         select read;
116       //var selectedReadings = from read in this.EnergyWindows
117       //                       where read.id == 10  select read;
118       int count = selectedReadings.Count();
119       foreach (var item in selectedReadings)
120       {
121         dbwindows.Add(item);
122       }
123       foreach (var item in dbwindows)
124       {
125         windows.Add(new EnergyWindow()
126         {
127           id = item.id,
128           Meter = meters[item.meter_id],
129           start_time = item.start_time,
130           end_time=item.end_time,
131           energy_consumed = item.energy_consumed,
132           // is_estimate = item.is_estimate,
133           is_fully_populated = item.is_fully_populated
134         });
135       }
```

```
136       }

137       //public double GetPreviousPeakValueForMeter(int meterID, DateTime hour)
138       //{ }
139       private void GetWindowsForAllMetersBasedOnTime(int hour, int minute, List<
            DbEnergyWindow> dbwindows, List<EnergyWindow> windows)
140       {
141         TimeSpan ts = new TimeSpan(hour, minute, 0);

142         DateTime time = DateTime.Now.Subtract(ts);
143         var selectedReadings = from read in this.EnergyWindows
144           where read.start_time > time
145           select read;

146         int count = selectedReadings.Count();
147         foreach (var item in selectedReadings)
148         {
149           dbwindows.Add(item);
150         }
151         foreach (var item in dbwindows)
152         {
153           windows.Add(new EnergyWindow()
154           {
155             id = item.id,
156             Meter = meters[item.meter_id],
157             start_time = item.start_time,
158             energy_consumed = item.energy_consumed,
159             // is_estimate = item.is_estimate,
160             is_fully_populated = item.is_fully_populated
161           });
162         }
163       }

164     }
165   }
```

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;

 5  namespace Models
 6  {
 7    public class EnergyReading
 8    {
 9      public int id {
10      }

11      public int meter_id {
12      }

13      public bool is_estimate {
14      }

15      public float value {
16      }

17      public DateTime measurement_timestamp {
18      }

19      public string measurement_unit {
20      }

21      public Meter Meter {
22      }

23      public double EstimatedCost
24      {
25        get
26        {
27          return value * pricePerKwhr;
28        }
29      }

30      public double pricePerKwhr {
31      }

32      public string Month
33      {
34          get { return measurement_timestamp.ToString("MMM"); }
35      }

36      public string Day
37      {
38          get { return measurement_timestamp.ToString("ddd"); }
39      }

40      public EnergyReading()
41      {
42        pricePerKwhr = 1.34;
43      }
```

```
44     }
45 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using Models;
6  using System.IO;
7  using Models.Enumerations;

8  namespace Dashboard_Control
9  {
10   /// <summary>
11   /// Class to read baseline files
12   /// </summary>
13   public class BaselineFileReader
14   {
15     private StreamReader reader;
16     private string filePathName;
17     private int CurrentLineNumber = -1;

18     /// <summary>
19     /// Constructor
20     /// </summary>
21     public BaselineFileReader()
22     {
23       filePathName = @"BaselineFile\baselineVariables.csv";
24       reader = new StreamReader(filePathName);
25     }
```

```
26     public List<BaselineVariable> GetVariables()
27     {
28       List<BaselineVariable> variables = new List<BaselineVariable>();
29       while (!reader.EndOfStream)
30       {

31         CurrentLineNumber++;
32         string row = reader.ReadLine();
33         if (!String.IsNullOrWhiteSpace(row))
34         {
35           string[] values = row.Split(',');
36           if (CurrentLineNumber == 0) // this is the header line.
37           {
38             string[] headerFields = row.Split(',');

39             if (headerFields.Length != 5)
40             {
41               reader.Close();
42               throw new InvalidOperationException("Header line \"" +
                   CurrentLineNumber + "\" does not have correct number of
                   fields.It has " + headerFields.Length + "headers");
43             }
44             return GetVariables();
45           }

46           BaselineVariable baselineVariable = new BaselineVariable();
47           int cycle = 0;
```

```
48          if (!int.TryParse(values[0], out cycle))
49          {
50            throw new InvalidOperationException("The baseline on  \"" +
                CurrentLineNumber + "\" must be a decimal!");

51          }
52          else
53            baselineVariable.HalfHourCycle = cycle;


54          string status = values[1];
55          switch (status)
56          {
57            case "Campus Closed":
58              baselineVariable.CampusStatus = eCampusStatus.CampusClosed;
59              break;
60            case "Special":
61              baselineVariable.CampusStatus = eCampusStatus.Special;
62              break;
63            case "Campus Open":
64              baselineVariable.CampusStatus = eCampusStatus.CampusOpen;
65              break;
66            default:
67              baselineVariable.CampusStatus = eCampusStatus.CampusOpen;
68              break;
69          }
70          string weekDay = values[2];
71          switch (weekDay)
72          {
73            case "Sunday":
74              baselineVariable.WeekDay = DayOfWeek.Sunday;
75              break;
76            case "Monday":
77              baselineVariable.WeekDay = DayOfWeek.Monday;
78              break;
79            case "Tuesday":
80              baselineVariable.WeekDay = DayOfWeek.Tuesday;
81              break;
82            case "Wednesday":
83              baselineVariable.WeekDay = DayOfWeek.Wednesday;
84              break;
85            case "Thursday":
86              baselineVariable.WeekDay = DayOfWeek.Thursday;
87              break;
88            case "Friday":
89              baselineVariable.WeekDay = DayOfWeek.Friday;
90              break;
91            case "Saturday":
92              baselineVariable.WeekDay = DayOfWeek.Saturday;
93              break;
94            default:
95              baselineVariable.WeekDay = DayOfWeek.Monday;
96              break;
97          }
```

```
 98          string month = values[3];
 99          switch (month)
100          {
101              case "January":
102                  baselineVariable.Month = 1;
103                  break;
104              case "February":
105                  baselineVariable.Month = 2;
106                  break;
107              case "March":
108                  baselineVariable.Month = 3;
109                  break;
110              case "April":
111                  baselineVariable.Month = 4;
112                  break;
113              case "May":
114                  baselineVariable.Month = 5;
115                  break;
116              case "June":
117                  baselineVariable.Month = 6;
118                  break;
119              case "July":
120                  baselineVariable.Month = 7;
121                  break;
122              case "August":
123                  baselineVariable.Month = 8;
124                  break;
125              case "September":
126                  baselineVariable.Month = 9;
127                  break;
128              case "October":
129                  baselineVariable.Month = 10;
130                  break;
131              case "November":
132                  baselineVariable.Month = 11;
133                  break;
134              case "December":
135                  baselineVariable.Month = 12;
136                  break;
137              default:
138                  baselineVariable.Month = 6;
139                  break;
140          }
141          float baseline = 0;
142          if (!float.TryParse(values[4], out baseline))
143          {
144              reader.Close();
145              throw new InvalidOperationException("The baseline on  \"" +
                     CurrentLineNumber + "\" must be a decimal!");
146          }
147          else
148              baselineVariable.BaselineValue = baseline;
```

```
149             variables.Add(baselineVariable);
150         }
151     }
152     reader.Close();
153     return variables;
154   }
155 }
156 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;

5  namespace Models
6  {
7    public class Location
8    {
9      public int id {
10     }

11     public string name {
12     }

13     public List <Meter> Meters {
14     }

15     public Location()
16     {
17       Meters = new List<Meter>();
18     }

19   }
20 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;

5  namespace Models
6  {
7    public class EnergyMeasurement
8    {
9      public int id {
10     }

11     public int meter_id {
12     }

13     public bool is_Estimate {
14     }

15     public float value {
16     }

17     public DateTime measurement_timestamp {
18     }

19     public Meter Meter {
20     }
21   }
22 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;

5  namespace Models
6  {
7    public class Meter
8    {
9      public int id {
10     }

11     public string serialno {
12     }

13     public int location_id {
14     }

15     public int meter_status_id {
16     }

17     // public Location Location { get; set; }
18     public string Location {
19     }

20     public MeterStatus MeterStatus {
21     }

22     public bool IsShown {
23     }

24     public Meter()
25     {
26       IsShown = true;
27     }

28   }
29 }
```

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Windows.Forms;
 5  using System.Reflection;
 6  using System.IO;

 7  namespace EnergyManagementDashboard
 8  {
 9    static class Program
10    {
11      /// <summary>
12      /// The main entry point for the application.
13      /// </summary>
14      static void Main()
15      {
16        Application.EnableVisualStyles();
17        Application.SetCompatibleTextRenderingDefault(false);
18        //
19        string fileName = "DevExpress.BonusSkins.v12.2.dll";

20        if (File.Exists(fileName))
21        {

22          // Assembly SampleAssembly =
    Assembly.LoadFile(Path.GetFullPath(fileName));

23          System.Reflection.Assembly SampleAssembly = System.Reflection.Assembly.
              LoadFrom("DevExpress.BonusSkins.v12.2.dll");
24          DevExpress.Skins.SkinManager.Default.RegisterAssembly(SampleAssembly);
25        }
26        //



27        DevExpress.LookAndFeel.UserLookAndFeel.Default.SkinName = "Glass Oceans";
    // <<< NEW LIN
28        Application.Run(new FrmLogin());
29      }

30    }
31  }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;

5  namespace Models.Enumerations
6  {
7    public enum eSelectionFormat
8    {
9      DateTime,
10     HoursMinutes
11   }
12 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;

5  namespace Models
6  {
7    public class MeterStatus
8    {
9      public int id {
10     }

11     public string name {
12     }
13   }
14 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Windows.Forms;
9  using Models;
10 using Configuration;

11 namespace EnergyManagementDashboard
12 {
13   public partial class FrmLogin : Form
14   {
15     public FrmLogin()
16     {
17       InitializeComponent();
18     }
```

```
19     private void btnLogin_Click(object sender, EventArgs e)
20     {
21       this.Hide();
22       ConfigSettings cS =SetUpDbConnectionSettings();
23       FrmMain fm = new FrmMain();
24       fm.LoadMainForm(cS);
25       fm.Show();
26     }
```

```
27     private ConfigSettings SetUpDbConnectionSettings()
28     {
29       Settings settings = new Settings();
30       ConfigSettings cS = new ConfigSettings();
31       if (!settings.DoesFileExist())
32       {
33         MessageBox.Show("Database Settings have not been set please input them.
           ,
34         "Wits Dashboard", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

35         FrmDBSettings fDbSettings = new FrmDBSettings();
36         fDbSettings.ShowDialog();
37       }
38       else
39       {
40         cS=settings.ReadSettings();
41       }

42       return cS;
43     }
```

```
44     private void btnExit_Click(object sender, EventArgs e)
45     {
46       Close();
```

```
47      }

48      private void FrmLogin_Load(object sender, EventArgs e)
49      {
50        SetUpUsers();
51      }

52      private void SetUpUsers()
53      {
54        User one = new User("Rodwell", "Genesis");
55        User two = new User("SuperUser", "none");
56        User thre = new User("Guest", "none");
57        BindingList<User> users = new BindingList<User>();
58        users.Add(one);
59        users.Add(two);
60        users.Add(thre);
61        gridLookUpEditUserNames.Properties.DataSource = users;
62        gridLookUpEditUserNames.EditValue = users[2];
63      }

64    }
65  }
```

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;

 5  namespace Models
 6  {
 7    public class EnergyWindow
 8    {
 9      public int id {
10      }

11      //  public int meter_id { get; set; }
12      public DateTime start_time {
13      }

14      public DateTime end_time {
15      }

16      public double energy_consumed {
17      }

18      public bool is_fully_populated {
19      }

20      // public bool? is_estimate { get{if; set; }
21      public Meter Meter {
22      }

23      public string StartTimeMonth
24      {
25          get { return start_time.ToString("MMM"); }
26      }

27      public string EndTimeMonth
28      {
29          get { return end_time.ToString("MMM"); }
30      }
31    }
32  }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;

5  namespace DataAccess
6  {
7    public class DbEnergyWindow
8    {
9      public int id {
10     }

11     public int meter_id {
12     }

13     public DateTime start_time {
14     }

15     public DateTime end_time {
16     }

17     public double energy_consumed {
18     }

19     public bool is_fully_populated {
20     }
21   }
22 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;

5  namespace DataAccess
6  {
7    public class DbEnergyMeasurement
8    {
9      public int id {
10     }

11     public int meter_id {
12     }

13     public bool is_Estimate {
14     }

15     public float value {
16     }

17     public DateTime measurement_timestamp {
18     }
19   }
20 }
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Models.Enumerations;
using Models;

namespace Dashboard_Control
{
  public class AlgorithmCalculations
  {
    /// <summary>
    /// List of all base line variables
    /// </summary>
    private List<BaselineVariable> AllBaselineVariables;

    public AlgorithmCalculations(List<BaselineVariable> list)
    {
      this.AllBaselineVariables = list;
    }
```

```
    public float getBaselineValue(int halfHourCycle, eCampusStatus
      campusOpenStatus, DayOfWeek dayOfWeek, int Month)
    {
      BaselineVariable baselineVariable = AllBaselineVariables.FirstOrDefault(x
        => x.HalfHourCycle == halfHourCycle && x.CampusStatus ==
        campusOpenStatus && x.WeekDay == dayOfWeek && x.Month == Month);

      if (baselineVariable == null)
      {
        if (campusOpenStatus == eCampusStatus.CampusClosed)
        {
          campusOpenStatus = eCampusStatus.CampusOpen;
        }
        else if (campusOpenStatus == eCampusStatus.CampusOpen)
        {
          campusOpenStatus = eCampusStatus.CampusClosed;
        }
        else if (campusOpenStatus == eCampusStatus.Special)
        {
          campusOpenStatus = eCampusStatus.CampusOpen;
        }
        return getBaselineValue(halfHourCycle,campusOpenStatus ,  dayOfWeek,
          Month);
      }
      return baselineVariable.BaselineValue;
    }
```

```
    public double CalculateForecastLoadForCycle(float temperature , float
      previousPeak,float baselineValue)
    {
      //loat predictedLoad = 0;
      double predictedLoad = -3.059e+00 + (9.967e-02 * temperature) + (6.241E-1
         * previousPeak) + (1.595E-04 * temperature * temperature * temperature
```

```
            + (4.378e-01 * baselineValue) - (3.566E-03 * temperature *
          baselineValue);

      return predictedLoad;
    }
  }
}
```

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;
 5  using System.Data.Entity;
 6  using Models;

 7  //using System.Data.Entity.ModelConfiguration.Conventions;
 8  //using System.Data.Entity.Migrations;
 9  namespace DataAccess
10  {
11    public class PowerReadingRepository : DbContext
12    {
13      public DbSet<DbPowerReading> Readings {
14      }
```

```
15      public PowerReadingRepository(Npgsql.NpgsqlConnection conn)
16      : base(conn, true)
17      {
18      }
```

```
19      protected override void OnModelCreating(DbModelBuilder modelBuilder)
20      {
21        modelBuilder.Entity<DbPowerReading>().ToTable("readings", "public");
22      }
```

```
23      /// <summary>
24      /// Load readings from the past x minutes
25      /// </summary>
26      /// <param name="elapsedTime"></param>
27      /// <returns></returns>
28      public List<DbPowerReading> LoadReadings(int elapsedTime,int meterID=
29    }
30  selectedReadings
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;

5  namespace DataAccess
6  {
7    public class DbMeterStatus
8    {
9      public int id {
10     }

11     public string name {
12     }
13   }
14 }
```

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;
 5  using System.Data.Entity.ModelConfiguration.Conventions;
 6  using System.ComponentModel.DataAnnotations.Schema;

 7  namespace DataAccess
 8  {
 9    [Table("meters", Schema = "public")]
10    public class DbMeter
11    {
12      public int id {
13      }

14      public string serialno {
15      }

16      //public int location_id { get; set; }
17      public int meter_status_id {
18      }

19      public string name {
20      }
21    }
22  }
```