

```
1 using WcfService.domain.order;
2 using System;
3 using System.Collections.Generic;
4 using System.IO;
5 using System.Linq;
6 using System.Net.Http;
7 using System.Runtime.Serialization;
8 using System.ServiceModel;
9 using System.ServiceModel.Web;
10 using System.Text;
11
12 namespace WcfService
13 {
14     [ServiceContract]
15     public interface IRestService
16     {
17         [OperationContract]
18         [WebInvoke(Method = "GET",
19             ResponseFormat = WebMessageFormat.Json,
20             BodyStyle = WebMessageBodyStyle.Wrapped,
21             UriTemplate = "getOrder/{orderNumber}")]
22         OrderConfirmation getOrder(string orderNumber);
23
24         [OperationContract]
25         [WebInvoke(Method="GET",
26             ResponseFormat = WebMessageFormat.Json,
27             BodyStyle = WebMessageBodyStyle.Wrapped,
28             UriTemplate = "getAllActiveOrders")]
29         List<OrderConfirmation> getAllActiveOrders();
30
31         [OperationContract]
32         [WebInvoke(Method = "GET",
33             ResponseFormat = WebMessageFormat.Json,
34             BodyStyle = WebMessageBodyStyle.Wrapped,
35             UriTemplate = "getUpdates")]
36         int getUpdates();
37
38
39         [OperationContract]
40         [WebInvoke(Method = "POST",
41             RequestFormat = WebMessageFormat.Json,
42             UriTemplate = "addOrderConfirmation")]
43         string addOrderConfirmation(List<string> fileContent);
44
45         [OperationContract]
46         [WebInvoke(Method = "POST",
47             UriTemplate = "addNote",
48             RequestFormat = WebMessageFormat.Json,
49             BodyStyle = WebMessageBodyStyle.Wrapped,
50             ResponseFormat = WebMessageFormat.Json)]
51         bool addNote(Stream stream);
52     }
53 }
54
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Runtime.Serialization;
5 using System.ServiceModel;
6 using System.Text;
7 using WcfService.domain.order;
8 using System.Net.Http;
9 using System.IO;
10 using System.Web;
11 using System.Net;
12 using System.Web.Hosting;
13 using System.ServiceModel.Web;
14 using WcfService.domain.data;
15 using WcfService.technical;
16
17 namespace WcfService
18 {
19     public class RestService : IRestService
20     {
21         public static int Updates = 0;
22
23         /// <summary>
24         /// Gets an order by the orders number
25         /// </summary>
26         /// <param name="orderNumber"></param>
27         /// <returns></returns>
28         OrderConfirmation IRestService.getOrder(string orderNumber)
29         {
30             return DBHandler.Instance.getOrder(orderNumber);
31         }
32
33         /// <summary>
34         /// Tries to create a order confirmation file from a given string array,
35         /// which then is parsed into an object by the OrderParser and stored on the DB.
36         /// </summary>
37         /// <param name="fileContent">String array of each line in the file, first line given is the filename</param>
38         /// <returns></returns>
39         public string addOrderConfirmation(List<string> fileContent)
40         {
41             try
42             {
43                 string fileName = fileContent[0].Split('.')[0];
44                 string fileExtension = fileContent[0].Split('.')[1];
45                 FileInfo fileInfo = new FileInfo(Path.Combine
46                     (HostingEnvironment.MapPath("~/Order/" + fileName + "/"),
47                     fileName + "." + fileExtension));
48
49                 if (fileInfo.Exists)
50                     return "ERROR: File already exists.";
51
52                 fileInfo.Directory.Create();
53                 using (var output = new StreamWriter(File.Create
```

```
(fileInfo.FullName), Encoding.Default))
52     {
53         for (int i = 1; i < fileContent.Count; i++)
54         {
55             output.WriteLine(fileContent[i]);
56         }
57         output.Flush();
58         output.Close();
59     }
60
61     OrderConfirmation oc = OrderParser.Instance.readOrder
62         (fileInfo.FullName);
63
64     DBHandler.Instance.createOrder(oc);
65 }
66 catch (IOException)
67 {
68     return "ERROR: File was not uploaded correctly.";
69 }
70 Updates++;
71 return "OK: File was successfully uploaded to the service.";
72 }
73
74 /// <summary>
75 /// Returns all the active orders from the Database
76 /// </summary>
77 /// <returns>List of OrderConfirmation</returns>
78 public List<OrderConfirmation> getAllActiveOrders()
79 {
80     return DBHandler.Instance.getAllOrdersOfStatus("Active");
81 }
82
83 public int getUpdates()
84 {
85     return Updates;
86 }
87
88 /// <summary>
89 /// Saves the note from the Android and stores it in the Database
90 /// </summary>
91 /// <param name="stream">Stream containing note information</param>
92 /// <returns></returns>
93 public bool addNote(Stream stream)
94 {
95     string dataText = "";
96     try
97     {
98         StreamReader sr = new StreamReader(stream, Encoding.UTF8);
99
100         dataText = sr.ReadToEnd().Replace("\\"", "");
101         int index = dataText.IndexOf(';');
102         string orderNumber = dataText.Substring(0, index).Trim();
103         string value = dataText.Substring(index + 1).Trim();
104
105         DBHandler.Instance.createNotes(orderNumber, new OrderNote
106             (value));
```

```
105         }
106         catch (Exception)
107         {
108             return false;
109         }
110         Updates++;
111         return true;
112     }
113 }
114 }
115
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Runtime.Serialization;
5 using System.ServiceModel;
6 using System.ServiceModel.Web;
7 using System.Text;
8 using System.Web.UI;
9 using WcfService1.domain.order;
10
11 namespace WcfService1
12 {
13     // NOTE: You can use the "Rename" command on the "Refactor" menu to change  
14     // the interface name "IServiceWGet" in both code and config file together.
15     [ServiceContract]
16     public interface IServiceWGet
17     {
18         [OperationContract]
19         [WebInvoke(Method = "GET",
20             ResponseFormat = WebMessageFormat.Json,
21             BodyStyle = WebMessageBodyStyle.Wrapped,
22             UriTemplate = "getUpdates")]
23         int getUpdates();
24
25         [OperationContract]
26         [WebInvoke(Method = "GET",
27             ResponseFormat = WebMessageFormat.Json,
28             BodyStyle = WebMessageBodyStyle.Wrapped,
29             UriTemplate = "getAllActiveOrders")]
30         List<OrderConfirmation> getAllActiveOrders();
31     }
32 }
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.IO;
4 using System.Linq;
5 using System.Runtime.Serialization;
6 using System.ServiceModel;
7 using System.Text;
8 using System.Web.Hosting;
9 using WcfService;
10 using WcfService.domain.data;
11 using WcfService.domain.order;
12 using WcfService.technical;
13
14 namespace WcfService1
15 {
16     // NOTE: You can use the "Rename" command on the "Refactor" menu to change ↗
17     // the class name "ServiceWGet" in code, svc and config file together. ↗
18     // NOTE: In order to launch WCF Test Client for testing this service, ↗
19     // please select ServiceWGet.svc or ServiceWGet.svc.cs at the Solution ↗
20     // Explorer and start debugging.
21     public class ServiceWGet : IServiceWGet
22     {
23         public List<OrderConfirmation> getAllActiveOrders()
24         {
25             return DBHandler.Instance.getAllOrdersOfStatus("Active");
26         }
27
28         public int getUpdates()
29         {
30             return RestService.Updates;
31         }
32     }
33 }
```

```
1 using WcfService.domain.order;
2 using System;
3 using System.Collections.Generic;
4 using System.Data.SqlClient;
5 using System.Data;
6
7 namespace WcfService.technical
8 {
9     class DBHandler
10    {
11
12        private static DBHandler instance;
13        private string dbName;
14        private string dbTable;
15        private string connectionString;
16
17        /// <summary>
18        /// Constructs a Singleton
19        /// </summary>
20        public static DBHandler Instance
21        {
22            get
23            {
24                if (instance == null) instance = new DBHandler();
25                return instance;
26            }
27            private set
28            {
29                instance = value;
30            }
31        }
32
33        /// <summary>
34        /// Reads the Properties File for the Connection information
35        /// </summary>
36        private DBHandler() {
37            domain.Properties prop = new domain.Properties("database");
38            prop.reload();
39            if (!prop.keyExists("databaseName"))
40                prop.set("databaseName", "localhost");
41            if (!prop.keyExists("databaseTable"))
42                prop.set("databaseTable", "SKDB");
43            prop.Save();
44            dbName = prop.get("databaseName");
45            dbTable = prop.get("databaseTable");
46            connectionString = String.Format("Data Source={0};Initial Catalog=
47                {1};Integrated Security=True", dbName, dbTable);
48        }
49
50        // Create Methods
51
52        /// <summary>
53        /// Recieves the OrderConfirmation Object,
54        /// which then is used create an order on the database.
55        /// </summary>
56        /// <param name="orderConfirmation"></param>
```

```
56     public void createOrder(OrderConfirmation orderConfirmation)
57     {
58         try
59         {
60             using (SqlConnection connection = new SqlConnection      ↗
61                 (connectionString))
62             {
63                 connection.Open();
64                 using (SqlCommand command = new SqlCommand("createOrder", ↗
65                     connection) { CommandType = CommandType.StoredProcedure })
66                 {
67                     //Parsing arrays of strings to a complete string
68                     string di = "";
69                     foreach (string str in orderConfirmation.DeliveryInfo)
70                         di += str + ";";
71                     di.Substring(0, di.Length - 1);
72
73                     string adi = "";
74
75                     foreach (string str in      ↗
76                         orderConfirmation.AltDeliveryInfo)
77                         adi += str + ";";
78                     adi.Substring(0, adi.Length - 1);
79
80                     command.Parameters.Add(new SqlParameter      ↗
81                         ("@OrderNumber", SqlDbType.VarChar, 64));
82                     command.Parameters["@OrderNumber"].Value =      ↗
83                         orderConfirmation.OrderNumber;
84                     command.Parameters.Add(new SqlParameter("@OrderName", ↗
85                         SqlDbType.VarChar, 64));
86                     command.Parameters["@OrderName"].Value =      ↗
87                         orderConfirmation.OrderName;
88                     command.Parameters.Add(new SqlParameter("@Delivery", ↗
89                         SqlDbType.VarChar, 256));
90                     command.Parameters["@Delivery"].Value = di;
91                     command.Parameters.Add(new SqlParameter      ↗
92                         ("@AltDelivery", SqlDbType.VarChar, 256));
93                     command.Parameters["@AltDelivery"].Value = adi;
94                     command.Parameters.Add(new SqlParameter      ↗
95                         ("@HousingAssociation", SqlDbType.VarChar, 64));
96                     command.Parameters["@HousingAssociation"].Value =      ↗
97                         orderConfirmation.HousingAssociation;
98                     command.Parameters.Add(new SqlParameter("@StartDate", ↗
99                         SqlDbType.Date));
100                     command.Parameters["@StartDate"].Value =      ↗
101                         orderConfirmation.ProducedDate.ToString("yyyy-MM-dd");
102                     command.Parameters.Add(new SqlParameter      ↗
103                         ("@DeliveryDate", SqlDbType.Date));
104                     command.Parameters["@DeliveryDate"].Value =      ↗
105                         orderConfirmation.OrderDate.ToString("yyyy-MM-dd");
106                     command.Parameters.Add(new SqlParameter      ↗
107                         ("@DeliveryWeek", SqlDbType.VarChar, 32));
108                     command.Parameters["@DeliveryWeek"].Value =      ↗
109                         orderConfirmation.Week;
110                     command.Parameters.Add(new SqlParameter      ↗
111                         ("@BlueprintLink", SqlDbType.VarChar, 256));
```



```

94         command.Parameters["@BlueprintLink"].Value = "";
95         command.Parameters.Add(new SqlParameter
96             ("@RequisitionLink", SqlDbType.VarChar, 256));
97         command.Parameters.Add(new SqlParameter
98             ("@ProgressStatus", SqlDbType.VarChar, 16));
99         command.Parameters["@ProgressStatus"].Value =
orderConfirmation.Status;
100         command.Parameters.Add(new SqlParameter("@CompanyID",
101             SqlDbType.Int));
102         command.Parameters["@CompanyID"].Value = 1;
103         command.ExecuteNonQuery();
104     }
105     connection.Close();
106
107     foreach (OrderCategory category in
orderConfirmation.Categories)
108     {
109         createOrderCategory(orderConfirmation.OrderNumber,
110             category);
111     }
112 }
113 catch (SqlException ex)
114 {
115     Console.WriteLine(ex.Message);
116     Console.WriteLine(ex.StackTrace);
117     throw;
118 }
119
120 /// <summary>
121 /// Stores a note on the database, linked to a given OrderNumber.
122 /// </summary>
123 /// <param name="orderNumber">OrderNumber of an order</param>
124 /// <param name="note">Note to store</param>
125 /// <returns></returns>
126 public List<OrderNote> createNotes(string orderNumber, OrderNote note)
127 {
128     List<OrderNote> result = new List<OrderNote>();
129     try
130     {
131         using (SqlConnection connection = new SqlConnection
132             (connectionString))
133         {
134             connection.Open();
135             using (SqlCommand command = new SqlCommand("createNotes",
136                 connection) { CommandType = CommandType.StoredProcedure })
137             {
138                 command.Parameters.Add(new SqlParameter
139                     ("@OrderNumber", SqlDbType.VarChar, 64));
140                 command.Parameters["@OrderNumber"].Value =
orderNumber;
141                 command.Parameters.Add(new SqlParameter("@Content",
142                     SqlDbType.VarChar, 1024));

```

```
139         command.Parameters["@Content"].Value = note.Text;
140         command.ExecuteNonQuery();
141     }
142     connection.Close();
143 }
144 }
145 catch (SqlException ex)
146 {
147     Console.WriteLine(ex.Message);
148 }
149 return result;
150 }
151
152 /// <summary>
153 /// Stores a category from an order on the database, linked to a given OrderNumber.
154 /// </summary>
155 /// <param name="orderNumber">OrderNumber of an order</param>
156 /// <param name="category">Category to store</param>
157 /// <returns></returns>
158 public List<OrderCategory> createOrderCategory(string orderNumber, OrderCategory category)
159 {
160     List<OrderCategory> result = new List<OrderCategory>();
161     try
162     {
163         using (SqlConnection connection = new SqlConnection \(connectionString\))
164         {
165             int categoryID = -1;
166             connection.Open();
167             using (SqlCommand command = new SqlCommand \("createOrderCategory", connection\) { CommandType = CommandType.StoredProcedure })
168             {
169                 command.Parameters.Add(new SqlParameter \("@OrderNumber", SqlDbType.VarChar, 64\));
170                 command.Parameters["@OrderNumber"].Value = orderNumber;
171                 command.Parameters.Add(new SqlParameter \("@CategoryName", SqlDbType.VarChar, 64\));
172                 command.Parameters["@CategoryName"].Value = category.Name;
173                 //command.Parameters.Add(new SqlParameter("@new_id", SqlDbType.Int, 0, "fldCategoryID"\).Direction = ParameterDirection.Output);
174                 command.ExecuteNonQuery();
175
176                 //categoryID = Convert.ToInt32(command.Parameters \["@new\_id"\].Value);
177             }
178             connection.Close();
179
180             //Gets the last ID created, in this case the id for this Category
181             categoryID = getLastID("dbo.TblOrderCategory");
```

```
182
183         //Creates all the elements for the category
184         foreach (OrderElement element in category.Elements)
185         {
186             createOrderElements(element, categoryID);
187         }
188     }
189 }
190 catch (SqlException ex)
191 {
192     Console.WriteLine(ex.Message);
193 }
194 return result;
195 }
196
197 /// <summary>
198 /// Stores a element from a category on the database, linked to a given CategoryID.
199 /// </summary>
200 /// <param name="element">Element to store</param>
201 /// <param name="categoryId">CategoryID to identify which category the element is linked to</param>
202 public void createOrderElements(OrderElement element, int categoryId)
203 {
204     try
205     {
206         using (SqlConnection connection = new SqlConnection
207             (connectionString))
208         {
209             connection.Open();
210             using (SqlCommand command = new SqlCommand
211                 ("createOrderElements", connection) { CommandType =
212                     CommandType.StoredProcedure })
213             {
214                 command.Parameters.Add(new SqlParameter("@Pos",
215                     SqlDbType.VarChar, 32));
216                 command.Parameters["@Pos"].Value = element.Position;
217                 command.Parameters.Add(new SqlParameter("@Hinge",
218                     SqlDbType.VarChar, 64));
219                 command.Parameters["@Hinge"].Value = element.Hinge;
220                 command.Parameters.Add(new SqlParameter("@Finish",
221                     SqlDbType.VarChar, 64));
222                 command.Parameters["@Finish"].Value = element.Finish;
223                 command.Parameters.Add(new SqlParameter("@Amount",
224                     SqlDbType.VarChar, 32));
225                 command.Parameters["@Amount"].Value = element.Amount;
226                 command.Parameters.Add(new SqlParameter("@Unit",
227                     SqlDbType.VarChar, 32));
228                 command.Parameters["@Unit"].Value = element.Unit;
229
230                 string info = "";
231                 foreach (string str in element.ElementInfo)
232                 {
233                     info += str + ";";
234                 }
235             }
236         }
237     }
238 }
```

```

... \SK-4Sem\C#\SKServer\WcfService1\Technical\DBHandler.cs 6
228         command.Parameters.Add(new SqlParameter("@Text",  ↗
        SqlDbType.VarChar, 256));
229         command.Parameters["@Text"].Value = info;
230         command.Parameters.Add(new SqlParameter  ↗
        ("@Station4Status", SqlDbType.Bit));
231         command.Parameters["@Station4Status"].Value =  ↗
        element.StationStatus[0];
232         command.Parameters.Add(new SqlParameter  ↗
        ("@Station5Status", SqlDbType.Bit));
233         command.Parameters["@Station5Status"].Value =  ↗
        element.StationStatus[1];
234         command.Parameters.Add(new SqlParameter  ↗
        ("@Station6Status", SqlDbType.Bit));
235         command.Parameters["@Station6Status"].Value =  ↗
        element.StationStatus[2];
236         command.Parameters.Add(new SqlParameter  ↗
        ("@Station7Status", SqlDbType.Bit));
237         command.Parameters["@Station7Status"].Value =  ↗
        element.StationStatus[3];
238         command.Parameters.Add(new SqlParameter  ↗
        ("@Station8Status", SqlDbType.Bit));
239         command.Parameters["@Station8Status"].Value =  ↗
        element.StationStatus[4];
240         command.Parameters.Add(new SqlParameter("@CategoryID",  ↗
        SqlDbType.Int));
241         command.Parameters["@CategoryID"].Value = categoryId;
242
243         command.ExecuteNonQuery();
244     }
245     connection.Close();
246 }
247 }
248 catch (SqlException ex)
249 {
250     Console.WriteLine(ex.Message);
251 }
252 }
253
254 // Get methods
255
256 /// <summary>
257 /// Get an order from the Database by OrderNumber.
258 /// This get method also internally gets the order's categories with  ↗
    elements.
259 /// </summary>
260 /// <param name="orderNumber">OrderNumber of the order</param>
261 /// <returns></returns>
262 public OrderConfirmation getOrder(string orderNumber)
263 {
264     OrderConfirmation orderConfirmation = null;
265     try
266     {
267         using (SqlConnection connection = new SqlConnection  ↗
            (connectionString))
268         {
269             connection.Open();

```

```

... \SK-4Sem\C#\SKServer\WcfService1\Technical\DBHandler.cs 7
270 using (SqlCommand command = new SqlCommand("getOrder", 7
connection) { CommandType = CommandType.StoredProcedure })
271 {
272     command.Parameters.Add(new SqlParameter 7
("@OrderNumber", SqlDbType.VarChar, 64));
273     command.Parameters["@OrderNumber"].Value = 7
orderNumber;

274
275     using (SqlDataReader dr = command.ExecuteReader())
276     {
277         if (dr.HasRows)
278         {
279             while (dr.Read())
280             {
281                 orderConfirmation = new OrderConfirmation 7
();
282                 orderConfirmation.OrderNumber = 7
dr.GetString(0); //Order Number
283                 orderConfirmation.OrderName = dr.GetString 7
(1); //Order Name
284
285                 string[] splitName = 7
orderConfirmation.OrderName.Split('/');
286                 orderConfirmation.AlternativeNumber 7
= //Alternative Number
287                 splitName[0] + " " +
288                 orderConfirmation.OrderNumber + " " +
289                 splitName[1];
290
291                 orderConfirmation.ProducedDate = 7
dr.GetDateTime(2); //Start date/produced date
292                 orderConfirmation.OrderDate = 7
dr.GetDateTime(3); //Delivery date/order date
293                 orderConfirmation.Week = dr.GetString 7
(4); //Delivery week
294                 foreach (string line in dr.GetString 7
(5).Split(';')) //Delivery info
295                     orderConfirmation.DeliveryInfo.Add 7
(line);
296                 foreach (string line in dr.GetString 7
(6).Split(';')) //Alt delivery info
297                     orderConfirmation.AltDeliveryInfo.Add 7
(line);
298                 orderConfirmation.HousingAssociation = 7
dr.GetString(7); //Housing Association
299                 orderConfirmation.Status = dr.GetString 7
(10); //Status
300
301                 //Company info:
302                 orderConfirmation.CompanyInfo.Add 7
(dr.GetString(11)); //Name
303                 orderConfirmation.CompanyInfo.Add 7
(dr.GetString(12)); //Address
304                 orderConfirmation.CompanyInfo.Add 7
(dr.GetString(13)); //ZipCode
305                 orderConfirmation.CompanyInfo.Add 7

```

```

306         (dr.GetString(14));    //Phone
                                orderConfirmation.CompanyInfo.Add
307         (dr.GetString(15));    //FaxPhone
                                orderConfirmation.CompanyInfo.Add
308         (dr.GetString(16));    //CVR
                                orderConfirmation.CompanyInfo.Add
309         (dr.GetString(17));    //Email
                                }
310         }
311     }
312 }
313 connection.Close();
314
315 //Adds all the categories linked to this order confirmation
316 foreach (OrderCategory category in getOrderCategories
317 (orderNumber))
318 {
319     orderConfirmation.Categories.Add(category);
320 }
321 foreach (OrderNote note in getNotes(orderNumber))
322 {
323     orderConfirmation.Notes.Add(note);
324 }
325 }
326 catch (Exception ex)
327 {
328     Console.WriteLine(ex.StackTrace);
329 }
330 return orderConfirmation;
331 }
332
333 /// <summary>
334 /// Get a list of Order Confirmations that are active.
335 /// This get method also internally gets all the orders categories
336 with elements.
337 /// </summary>
338 /// <param name="status"></param>
339 /// <returns></returns>
340 public List<OrderConfirmation> getAllOrdersOfStatus(string status)
341 {
342     List<OrderConfirmation> result = new List<OrderConfirmation>();
343     try
344     {
345         using (SqlConnection connection = new SqlConnection
346 (connectionString))
347         {
348             connection.Open();
349             using (SqlCommand command = new SqlCommand
350 ("getAllOrdersOfStatus", connection) { CommandType =
351 CommandType.StoredProcedure })
352             {
353                 command.Parameters.Add(new SqlParameter
354 ("@ProgressStatus", SqlDbType.VarChar, 64));
355                 command.Parameters["@ProgressStatus"].Value = status;

```

```

351
352         using (SqlDataReader dr = command.ExecuteReader())
353         {
354             if (dr.HasRows)
355             {
356                 while (dr.Read())
357                 {
358                     OrderConfirmation orderConfirmation = new OrderConfirmation();
359                     orderConfirmation.OrderNumber = dr.GetString(0);
360                     orderConfirmation.OrderName = dr.GetString(1);
361                     string[] splitName = orderConfirmation.OrderName.Split('/');
362                     orderConfirmation.AlternativeNumber = splitName[0] + " " + orderConfirmation.OrderNumber + " " + splitName[1];
363                     orderConfirmation.ProducedDate = dr.GetDateTime(2);
364                     orderConfirmation.OrderDate = dr.GetDateTime(3);
365                     orderConfirmation.Week = dr.GetString(4);
366                     foreach (string line in dr.GetString(5).Split(';'))
367                     {
368                         orderConfirmation.DeliveryInfo.Add(line);
369                     }
370                     foreach (string line in dr.GetString(6).Split(';'))
371                     {
372                         orderConfirmation.AltDeliveryInfo.Add(line);
373                     }
374                     orderConfirmation.HousingAssociation = dr.GetString(7);
375                     orderConfirmation.Status = dr.GetString(10);
376                     //Company info:
377                     orderConfirmation.CompanyInfo.Add(dr.GetString(11));
378                     orderConfirmation.CompanyInfo.Add(dr.GetString(12));
379                     orderConfirmation.CompanyInfo.Add(dr.GetInt32(13) + "");
380                     orderConfirmation.CompanyInfo.Add(dr.GetString(14));
381                     orderConfirmation.CompanyInfo.Add(dr.GetString(15));
382                     orderConfirmation.CompanyInfo.Add(dr.GetString(16));
383                     orderConfirmation.CompanyInfo.Add(dr.GetString(17));

```

```

386         result.Add(orderConfirmation);
387     }
388 }
389 }
390 connection.Close();
391 //Get all categories for all orders
392 foreach (OrderConfirmation orderConfirmation in result)
393 {
394     foreach (OrderCategory category in getOrderCategories(orderConfirmation.OrderNumber))
395     {
396         orderConfirmation.Categories.Add(category);
397     }
398     foreach (OrderNote note in getNotes(orderConfirmation.OrderNumber))
399     {
400         orderConfirmation.Notes.Add(note);
401     }
402 }
403 }
404 }
405 }
406 catch (Exception ex)
407 {
408     Console.WriteLine(ex.StackTrace);
409 }
410 return result;
411 }
412
413 /// <summary>
414 /// Gets a list of Notes by OrderNumber
415 /// </summary>
416 /// <param name="orderNumber">OrderNumber of the order</param>
417 /// <returns></returns>
418 public List<OrderNote> getNotes(string orderNumber)
419 {
420     List<OrderNote> result = new List<OrderNote>();
421     try
422     {
423         using (SqlConnection connection = new SqlConnection(connectionString))
424         {
425             connection.Open();
426             using (SqlCommand command = new SqlCommand("getNotes", connection) { CommandType = CommandType.StoredProcedure })
427             {
428                 command.Parameters.Add(new SqlParameter("@OrderNumber", SqlDbType.VarChar, 64));
429                 command.Parameters["@OrderNumber"].Value = orderNumber;
430
431                 using (SqlDataReader dr = command.ExecuteReader())
432                 {
433                     if (dr.HasRows)
434
```



```
435         while (dr.Read())
436         {
437             result.Add(new OrderNote(dr.GetString
438                 (2)));
439         }
440     }
441 }
442 connection.Close();
443 }
444 }
445 catch (Exception ex)
446 {
447     Console.WriteLine(ex.StackTrace);
448 }
449 return result;
450 }
451
452 /// <summary>
453 /// Creates a list of Categories by orderNumber
454 /// </summary>
455 /// <param name="orderNumber">OrderNumber of the order</param>
456 /// <returns></returns>
457 public List<OrderCategory> getOrderCategories(string orderNumber)
458 {
459     List<OrderCategory> result = new List<OrderCategory>();
460     try
461     {
462         using (SqlConnection connection = new SqlConnection
463             (connectionString))
464         {
465             connection.Open();
466             using (SqlCommand command = new SqlCommand
467                 ("getCategories", connection) { CommandType =
468                     CommandType.StoredProcedure })
469             {
470                 command.Parameters.Add(new SqlParameter
471                     ("@OrderNumber", SqlDbType.VarChar, 64));
472                 command.Parameters["@OrderNumber"].Value =
473                     orderNumber;
474
475                 using (SqlDataReader dr = command.ExecuteReader())
476                 {
477                     if (dr.HasRows)
478                     {
479                         while (dr.Read())
480                         {
481                             OrderCategory category = new OrderCategory
482                                 (dr.GetString(1));
483                             result.Add(category);
484                         }
485                     }
486                 }
487             }
488             connection.Close();
489         }
490     }
491 }
```

```

484         for (int i = 0; i < result.Count; i++)
485         {
486             foreach (OrderElement element in getOrderElements(i  ➤
487                 +1))
488             {
489                 result[i].Elements.Add(element);
490             }
491         }
492     }
493     catch (Exception ex)
494     {
495         Console.WriteLine(ex.StackTrace);
496     }
497     return result;
498 }
499
500 /// <summary>
501 /// Creates a list of Elements by category ID
502 /// </summary>
503 /// <param name="categoryID">The CategoryID which the elements is  ➤
504   linked to</param>
505 /// <returns></returns>
506 public List<OrderElement> getOrderElements(int categoryID)
507 {
508     List<OrderElement> elements = new List<OrderElement>();
509     try
510     {
511         using (SqlConnection connection = new SqlConnection  ➤
512             (connectionString))
513         {
514             connection.Open();
515             using (SqlCommand command = new SqlCommand  ➤
516                 ("getOrderElements", connection) { CommandType =  ➤
517                     CommandType.StoredProcedure })
518             {
519                 command.Parameters.Add(new SqlParameter("@CategoryID",  ➤
520                     SqlDbType.VarChar, 64));
521                 command.Parameters["@CategoryID"].Value = categoryID;
522
523                 using (SqlDataReader dr = command.ExecuteReader())
524                 {
525                     if (dr.HasRows)
526                     {
527                         while (dr.Read())
528                         {
529                             OrderElement element = new OrderElement  ➤
530                                 (dr.GetString(1), dr.GetString(2), dr.GetString(3),  ➤
531                                 dr.GetString(4), dr.GetString(5)); //(dr.GetString(0),  ➤
532                                 dr.GetString(1), dr.GetString(2), dr.GetString(3),  ➤
533                                 dr.GetString(4));
534                             foreach (string str in dr.GetString  ➤
535                                 (6).Split(';'))
536                             {
537                                 element.ElementInfo.Add(str);
538                             }

```

```

529         element.StationStatus[0] = dr.GetBoolean
530             (7);
531         element.StationStatus[0] = dr.GetBoolean
532             (8);
533         element.StationStatus[0] = dr.GetBoolean
534             (9);
535         element.StationStatus[0] = dr.GetBoolean
536             (10);
537         element.StationStatus[0] = dr.GetBoolean
538             (11);
539         elements.Add(element);
540     }
541 }
542 catch (Exception ex)
543 {
544     Console.WriteLine(ex.StackTrace);
545 }
546 return elements;
547 }
548
549 /// <summary>
550 /// Retries the last created ID from a given table name.
551 /// </summary>
552 /// <param name="tableName">Table name of the desired ID</param>
553 /// <returns></returns>
554 public int getLastID(string tableName)
555 {
556     int id = -1;
557     try
558     {
559         using (SqlConnection connection = new SqlConnection
560             (connectionString))
561         {
562             connection.Open();
563             using (SqlCommand command = new SqlCommand(String.Format
564                 ("SELECT IDENT_CURRENT('{0}') AS ID", tableName),
565                 connection))
566             id = Convert.ToInt32(command.ExecuteScalar());
567             connection.Close();
568         }
569     }
570     catch (SqlException ex)
571     {
572         Console.WriteLine(ex.Message);
573     }
574     return id;
575 }

```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6 using System.IO;
7 using WcfService.domain.order;
8 using WcfService.domain.utility;
9
10 namespace WcfService.domain.data
11 {
12     public class OrderParser
13     {
14         private static OrderParser instance;
15
16         public static OrderParser Instance
17         {
18             get
19             {
20                 if (instance == null) instance = new OrderParser();
21                 return instance;
22             }
23             private set
24             {
25                 instance = value;
26             }
27         }
28
29         /// <summary>
30         /// Singleton constructor. This class can only be created from within.
31         /// </summary>
32         private OrderParser()
33         {
34
35         }
36
37         /// <summary>
38         ///
39         /// </summary>
40         /// <param name="url"></param>
41         public OrderConfirmation readOrder(string filePath)
42         {
43             OrderConfirmation result = new OrderConfirmation();
44             StreamReader stream = null;
45             try
46             {
47                 if (!filePath.ToLower().EndsWith(".e02"))
48                 {
49                     Console.Error.WriteLine("Error: Wrong file type.");
50                     return null;
51                 }
52
53                 if (!File.Exists(filePath))
54                 {
55                     Console.Error.WriteLine("Error: File doesn't exist.");
56                     return null;
57                 }
58             }
59             catch { }
60         }
61     }
62 }
```

```
57     }
58
59     stream = new StreamReader(filePath, Encoding.Default);
60
61     int kitchenInfoNumber = 0;
62     OrderElement lastElement = null;
63
64     while (stream.Peek() > -1)
65     {
66         string line = stream.ReadLine();
67         string[] lineSplit = line.Split(';');
68         string[] date;
69
70         if (lineSplit.Length == 0)
71             continue;
72
73
74         switch (lineSplit[0])
75         {
76             case "0": // Program header + produced date
77                 date = lineSplit[4].Split('/');
78                 string[] time = lineSplit[5].Split(':');
79
80                 DateTime dateTime = new DateTime(
81                     ConversionUtil.stringToInt(date[2]),
82                     ConversionUtil.stringToInt(date[1]),
83                     ConversionUtil.stringToInt(date[0]),
84                     ConversionUtil.stringToInt(time[0]),
85                     ConversionUtil.stringToInt(time[1]),
86                     ConversionUtil.stringToInt(time[2]));
87
88                 result.ProducedDate = dateTime;
89                 break;
90             case "101": // Company Info
91                 result.CompanyInfo.Add(lineSplit[1]);
92                 result.CompanyInfo.Add(lineSplit[3]);
93                 result.CompanyInfo.Add(lineSplit[4]);
94                 result.CompanyInfo.Add(lineSplit[5]);
95                 result.CompanyInfo.Add(lineSplit[20]);
96                 result.CompanyInfo.Add(lineSplit[24]);
97                 result.CompanyInfo.Add(lineSplit[25]);
98                 result.CompanyInfo.Add(lineSplit[30]);
99                 break;
100             case "211": // Delivery Info
101                 result.DeliveryInfo.Add(lineSplit[1]);
102                 result.DeliveryInfo.Add(lineSplit[3]);
103                 result.DeliveryInfo.Add(lineSplit[4]);
104                 result.DeliveryInfo.Add(lineSplit[5]);
105                 result.DeliveryInfo.Add(lineSplit[20]);
106                 result.DeliveryInfo.Add(lineSplit[24]);
107                 result.DeliveryInfo.Add(lineSplit[25]);
108                 break;
109             case "212": // Alternative Delivery Address
110                 result.AltDeliveryInfo.Add(lineSplit[1]);
111                 result.AltDeliveryInfo.Add(lineSplit[3]);
112                 result.AltDeliveryInfo.Add(lineSplit[4]);
```

```

113         result.AltDeliveryInfo.Add(lineSplit[5]);
114         result.AltDeliveryInfo.Add(lineSplit[11]);
115         result.AltDeliveryInfo.Add(lineSplit[19]);
116         result.AltDeliveryInfo.Add(lineSplit[20]);
117         result.AltDeliveryInfo.Add(lineSplit[24]);
118         result.AltDeliveryInfo.Add(lineSplit[25]);
119         result.AltDeliveryInfo.Add(lineSplit[26]);
120         break;
121     case "300": // Rightside Info
122         result.OrderNumber = lineSplit[1];
123         result.OrderName = lineSplit[3];
124         date = lineSplit[4].Split('/');
125         result.OrderDate = new DateTime(
126             ConversionUtil.stringToInt(date[2]),
127             ConversionUtil.stringToInt(date[1]),
128             ConversionUtil.stringToInt(date[0]), 12, 0, 0);
129         result.Week = lineSplit[7];
130
131         string[] splitName = result.OrderName.Split('/');
132         result.AlternativeNumber =
133             splitName[0] + " " +
134             result.OrderNumber + " " +
135             splitName[1];
136         break;
137     case "410": // Start of kitchen info
138         result.KitchenInfo.Add("KInfo-" +
139             kitchenInfoNumber);
140         kitchenInfoNumber++;
141         break;
142     case "423": // Title of the order part
143         result.KitchenInfo.Add(lineSplit[2]);
144         break;
145     case "424": // Text for the order part (423)
146         result.KitchenInfo.Add(lineSplit[2]);
147         break;
148     case "425": // Finish of the order part
149         result.KitchenInfo.Add(lineSplit[2]); // Title
150         result.KitchenInfo.Add(lineSplit[4]); //
151         Description
152         break;
153     case "430": // Order Category Info
154         result.Categories.Add(new OrderCategory(lineSplit
155             [2], ConversionUtil.stringToInt(lineSplit[1])));
156         break;
157     case "500": // Order Element
158         lastElement = new OrderElement(lineSplit[1], "",
159             "", lineSplit[9], lineSplit[10]);
160
161         int metaPos = ConversionUtil.stringToInt(lineSplit
162             [2]);
163         if (metaPos > 0)
164             lastElement.Position += "." + metaPos;
165
166         lastElement.ElementInfo.Add(lineSplit[3]);
167         lastElement.ElementInfo.Add(lineSplit[8]);

```

```
163         result.findCategoryByParserID
           (ConversionUtil.stringToInt(lineSplit[12])).Elements.Add
           (lastElement);
164         break;
165         case "501": // Order Element - Hinge + Finish
166             Console.WriteLine(line);
167             lastElement.Hinge = lineSplit[8];
168             lastElement.Finish = lineSplit[9];
169             break;
170         case "502": // Order Element - Extra info
171             lastElement.ElementInfo.Add(lineSplit[1]);
172             break;
173         default:
174             break;
175     }
176 }
177 }
178 catch (Exception)
179 {
180     return null;
181 }
182 finally
183 {
184     try
185     {
186         stream.Close();
187     }
188     catch (NullReferenceException)
189     { }
190 }
191
192 //Instantiate the order as Active
193 result.Status = "Active";
194 return result;
195     }
196 }
197 }
198
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.Serialization;
4
5 namespace WcfService.domain.order
6 {
7     [DataContract]
8     public class OrderConfirmation
9     {
10         [DataMember]
11         public string OrderNumber { get; set; }
12         [DataMember]
13         public string AlternativeNumber { get; set; }
14         [DataMember]
15         public string OrderName { get; set; }
16         [DataMember]
17         public string Week { get; set; }
18         [DataMember]
19         public string Status { get; set; }
20         [DataMember]
21         public DateTime OrderDate { get; set; }
22         [DataMember]
23         public DateTime ProducedDate { get; set; }
24         [DataMember]
25         public string HousingAssociation { get; set; }
26         [DataMember]
27         public List<string> CompanyInfo { get; private set; }
28         [DataMember]
29         public List<string> DeliveryInfo { get; private set; }
30         [DataMember]
31         public List<string> AltDeliveryInfo { get; private set; }
32         [DataMember]
33         public List<string> KitchenInfo { get; private set; }
34         [DataMember]
35         public List<OrderCategory> Categories { get; private set; }
36         [DataMember]
37         public List<OrderNote> Notes { get; private set; }
38         [DataMember]
39         public OrderStatus StationStatus { get; private set; }
40
41         /// <summary>
42         ///   instanciates the variables
43         /// </summary>
44         public OrderConfirmation()
45         {
46             OrderNumber = "";
47             AlternativeNumber = "";
48             OrderName = "";
49             Week = "";
50             Status = "";
51             HousingAssociation = "";
52             OrderDate = new DateTime();
53             ProducedDate = new DateTime();
54             CompanyInfo = new List<string>();
55             DeliveryInfo = new List<string>();
56             AltDeliveryInfo = new List<string>();
```



```
57         Categories = new List<OrderCategory>();
58         Notes = new List<OrderNote>();
59         KitchenInfo = new List<string>();
60         StationStatus = new OrderStatus();
61     }
62
63     /// <summary>
64     /// Finds the orderconfirmation with the given parserID
65     /// </summary>
66     /// <param name="parserID"></param>
67     /// <returns></returns>
68     public OrderCategory findCategoryByParserID(int parserID)
69     {
70         foreach (OrderCategory category in Categories)
71         {
72             if (category.ParserID == parserID)
73                 return category;
74         }
75         return null;
76     }
77 }
78 }
79
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Runtime.Serialization;
5 using System.Web;
6
7 namespace WcfService.domain.order
8 {
9     [DataContract]
10    public class OrderNote
11    {
12        [DataMember]
13        public string Text { get; set; }
14
15        public OrderNote(string text)
16        {
17            Text = text;
18        }
19    }
20 }
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Runtime.Serialization;
5 using System.Text;
6 using System.Threading.Tasks;
7
8 namespace WcfService.domain.order
9 {
10     [DataContract]
11     public class OrderCategory
12     {
13         [DataMember]
14         public string Name { get; set; }
15         /// <summary>
16         /// Parser ID is used to assemble the categories when reading the E02 ➤
17         /// file.
18         /// </summary>
19         public int ParserID { get; set; }
20         [DataMember]
21         public List<OrderElement> Elements { get; private set; }
22
23         public OrderCategory(string name)
24         {
25             Elements = new List<OrderElement>();
26             Name = name;
27         }
28         public OrderCategory(string name, int parserID)
29         {
30             Elements = new List<OrderElement>();
31             Name = name;
32             ParserID = parserID;
33         }
34     }
35 }
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Runtime.Serialization;
5 using System.Text;
6 using System.Threading.Tasks;
7
8 namespace WcfService.domain.order
9 {
10     [DataContract]
11     public class OrderElement
12     {
13         [DataMember]
14         public string Position { get; set; }
15         [DataMember]
16         public List<string> ElementInfo { get; private set; }
17         [DataMember]
18         public string Hinge { get; set; }
19         [DataMember]
20         public string Finish { get; set; }
21         [DataMember]
22         public string Amount { get; set; }
23         [DataMember]
24         public string Unit { get; set; }
25         [DataMember]
26         public bool[] StationStatus { get; set; }
27
28         public OrderElement(string position, string hinge, string finish,  ↗
29             string amount, string unit)
30         {
31             Position = position;
32             ElementInfo = new List<string>();
33             Hinge = hinge;
34             Finish = finish;
35             Amount = amount;
36             Unit = unit;
37             StationStatus = new bool[5];
38         }
39
40         public void updateStatus(int stationNumber, bool status)
41         {
42             StationStatus[stationNumber - 4] = status;
43         }
44
45         public bool allDone()
46         {
47             return StationStatus[0] && StationStatus[1] && StationStatus[2] &&  ↗
48                 StationStatus[3] && StationStatus[4];
49         }
50     }
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Runtime.Serialization;
5 using System.Web;
6
7 namespace WcfService.domain.order
8 {
9     [DataContract]
10    public class OrderStatus
11    {
12        [DataMember]
13        public string Station4 { get; set; }
14        [DataMember]
15        public string Station5 { get; set; }
16        [DataMember]
17        public string Station6 { get; set; }
18        [DataMember]
19        public string Station7 { get; set; }
20        [DataMember]
21        public string Station8 { get; set; }
22        [DataMember]
23        public bool Finished { get; set; }
24
25        public OrderStatus()
26        {
27            Station4 = "NotStartet";
28            Station5 = "NotStartet";
29            Station6 = "NotStartet";
30            Station7 = "NotStartet";
31            Station8 = "NotStartet";
32            Finished = false;
33        }
34    }
35 }
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.IO;
5 using System.Web.Hosting;
6
7 namespace WcfService.domain
8 {
9     public class Properties
10     {
11         private Dictionary<String, String> list;
12         private FileInfo configFile;
13         private static DirectoryInfo configFolder;
14         public int Count
15         {
16             get
17             {
18                 return list.Count;
19             }
20         }
21
22         public Properties(String file)
23         {
24             configFile = new FileInfo(Path.Combine(HostingEnvironment.MapPath ↗
25                 ("~/config/"), file + ".ini"));
26             // Sets the configFolder field if needed
27             if (configFolder == null)
28                 configFolder = configFile.Directory;
29             // Creates the configFolder if needed
30             if (!configFolder.Exists)
31                 configFolder.Create();
32
33             reload(file);
34         }
35
36         public String get(String field, String defValue)
37         {
38             return (get(field) == null) ? (defValue) : (get(field));
39         }
40
41         public String get(String field)
42         {
43             return (list.ContainsKey(field)) ? (list[field]) : (null);
44         }
45
46         public void set(String field, Object value)
47         {
48             if (!list.ContainsKey(field))
49                 list.Add(field, value.ToString());
50             else
51                 list[field] = value.ToString();
52         }
53
54         public bool keyExists(string key)
55         {
56             return list.Keys.Contains(key);
57         }
58     }
59 }
```

```
56     }
57
58     public void Save()
59     {
60         Save(this.configFile);
61     }
62
63     public void Save(String filename)
64     {
65         Save(new FileInfo(configFolder.FullName + "/" + filename + ".ini"));
66     }
67
68     public void Save(FileInfo fileInfo)
69     {
70         if (!fileInfo.Exists)
71             fileInfo.Create();
72
73         StreamWriter file = new StreamWriter(fileInfo.FullName);
74
75         foreach (String prop in list.Keys.ToArray())
76         {
77             if (!String.IsNullOrEmpty(list[prop]))
78                 file.WriteLine(prop + "=" + list[prop]);
79         }
80
81         file.Close();
82     }
83
84     public void reload()
85     {
86         reload(this.configFile);
87     }
88
89     public void reload(String filename)
90     {
91         reload(new FileInfo(configFolder.FullName + "/" + filename + ".ini"));
92     }
93
94     public void reload(FileInfo fileInfo)
95     {
96         list = new Dictionary<String, String>();
97
98         if (fileInfo.Exists)
99             loadFromFile(fileInfo);
100     else
101         fileInfo.Create();
102     }
103
104     private void loadFromFile(FileInfo fileInfo)
105     {
106         foreach (String line in File.ReadAllLines(fileInfo.FullName))
107         {
108             if ((!String.IsNullOrEmpty(line)) &&
109                 (!line.StartsWith(";"))) &&
```

```
110         (!line.StartsWith("#")) &&
111         (!line.StartsWith("'")) &&
112         (line.Contains('=')))
113     {
114         int index = line.IndexOf('=');
115         String key = line.Substring(0, index).Trim();
116         String value = line.Substring(index + 1).Trim();
117
118         if ((value.StartsWith("\'") && value.EndsWith("\'")) ||
119             (value.StartsWith("'") && value.EndsWith("'")))
120         {
121             value = value.Substring(1, value.Length - 2);
122         }
123
124         try
125         {
126             //ignore duplicates
127             list.Add(key, value);
128         }
129         catch { }
130     }
131 }
132 }
133
134
135 }
136 }
137
```