```
14
15      using System.Data.Entity;
16      using System.Xml.Linq;
17      using WATB.Datamodel.Factory;
18      using WATB.Datamodel.Generic;
19
20      namespace WATB.Datamodel.Context
21      {
25          public class GenericDatamodel : DbContext
26          {
30              public GenericDatamodel() : this("name=Datamodel") { }
31
36              public GenericDatamodel(string datamodel) : base(datamodel) { }
37
42              public virtual DbSet<Identifier> Identifiers { get; set; }
43
48              public virtual DbSet<DataContent> Contents { get; set; }
49
54              public virtual DbSet<DataTemplate> Templates { get; set; }
55
60              public virtual DbSet<BaseItem> Items { get; set; }
61
66              public virtual DbSet<ChildItem> Children { get; set; }
67
72              public virtual DbSet<Relation> Relations { get; set; }
73
78              public virtual DbSet<Property> Properties { get; set; }
79
84              public virtual DbSet<Asset> Assets { get; set; }
85
90              public virtual DbSet<BaseContract> Contracts { get; set; }
91
96              public virtual DbSet<Contract> OwnedContracts { get; set; }
97
98
99
104             public virtual DbSet<BaseLink> Links { get; set; }
105
110             public virtual DbSet<CountLink> CountLinks { get; set; }
111
116             public virtual DbSet<AmountLink> AmountLinks { get; set; }
117
124             public DataContent AddContent(XElement content, DataTemplate template) {
                 return Contents.Add(Contents.Create().Initialize(content, template)); }
125
132             public DataTemplate AddTemplate(XElement schema, XElement stylesheet) {
                return Templates.Add(Templates.Create().Initialize(schema, stylesheet));
                 }
133
141             public BaseItem AddItem(string name, XElement content = null,
                DataTemplate template = null) { return Items.Add(Items.Create().
                Initialize(name, content, template)); }
142
151             public ChildItem AddChild(string name, BaseItem parent, XElement content
                 = null, DataTemplate template = null) { return Children.Add(Children.
                Create().Initialize(name, parent, content, template)); }
152
161             public Relation AddRelation(string name, XElement content = null,
                DataTemplate template = null, params BaseItem[] items) { return
                Relations.Add(Relations.Create().Initialize(name, content, template,
                items)); }
162
172             public Property AddProperty(string name, BaseItem owner, XElement
    1   2
```

```
1 2
        content = null, DataTemplate template = null, params BaseItem[] items) {
         return Properties.Add(Properties.Create().Initialize(name, owner,
        content, template, items)); }
173
182     public Asset AddAsset(string name, XElement content = null, DataTemplate
         template = null, params BaseLink[] contracts) { return Assets.Add(
        Assets.Create().Initialize(name, content, template, contracts)); }
183
193     public BaseContract AddBaseContract(string name, ContractStatus status,
        XElement content = null, DataTemplate template = null, params BaseLink[]
         contracts) { return Contracts.Add(Contracts.Create().Initialize(name,
        status, content, template, contracts)); }
194
205     public Contract AddContract(string name, BaseItem owner, ContractStatus
        status, XElement content = null, DataTemplate template = null, params
        BaseLink[] contracts) { return OwnedContracts.Add(OwnedContracts.Create(
        ).Initialize(name, owner, status, content, template, contracts)); }
206
217     public BaseLink AddLink(LinkType linkType, BaseContract baseContract,
        BaseItem owner, Asset asset, XElement content = null, DataTemplate
        template = null) { return Links.Add(Links.Create().Initialize(linkType,
        baseContract, owner, asset, content, template)); }
218
230     public BaseLink AddLink(LinkType linkType, BaseContract baseContract,
        BaseItem owner, Asset asset, int count, XElement content = null,
        DataTemplate template = null) { return CountLinks.Add(CountLinks.Create(
        ).Initialize(linkType, baseContract, owner, asset, count, content,
        template)); }
231
243     public BaseLink AddLink(LinkType linkType, BaseContract baseContract,
        BaseItem owner, Asset asset, double amount, XElement content = null,
        DataTemplate template = null) { return AmountLinks.Add(AmountLinks.
        Create().Initialize(linkType, baseContract, owner, asset, amount,
        content, template)); }
244
254     public BaseLink AddSentLink(BaseContract baseContract, BaseItem owner,
        Asset asset, XElement content = null, DataTemplate template = null) {
        return Links.Add(Links.Create().Initialize(LinkType.Send, baseContract,
        owner, asset, content, template)); }
255
266     public BaseLink AddSentLink(BaseContract baseContract, BaseItem owner,
        Asset asset, int count, XElement content = null, DataTemplate template =
         null) { return CountLinks.Add(CountLinks.Create().Initialize(LinkType.
        Send, baseContract, owner, asset, count, content, template)); }
267
278     public BaseLink AddSentLink(BaseContract baseContract, BaseItem owner,
        Asset asset, double amount, XElement content = null, DataTemplate
        template = null) { return AmountLinks.Add(AmountLinks.Create().
        Initialize(LinkType.Send, baseContract, owner, asset, amount, content,
        template)); }
279
289     public BaseLink AddReceivedLink(BaseContract baseContract, BaseItem
        owner, Asset asset, XElement content = null, DataTemplate template =
        null) { return Links.Add(Links.Create().Initialize(LinkType.Receive,
        baseContract, owner, asset, content, template)); }
290
301     public BaseLink AddReceivedLink(BaseContract baseContract, BaseItem
        owner, Asset asset, int count, XElement content = null, DataTemplate
        template = null) { return CountLinks.Add(CountLinks.Create().Initialize(
        LinkType.Receive, baseContract, owner, asset, count, content, template))
        ; }
302
313     public BaseLink AddReceivedLink(BaseContract baseContract, BaseItem
        owner, Asset asset, double amount, XElement content = null, DataTemplate
1 2
```

```
                template = null) { return AmountLinks.Add(AmountLinks.Create().
               Initialize(LinkType.Receive, baseContract, owner, asset, amount, content
               , template)); }
314         }
315     }
```

```
14
15      using System;
16      using System.Collections.Generic;
17      using System.Linq;
18      using System.Xml.Linq;
19      using WATB.Datamodel.Generic;
20
21      namespace WATB.Datamodel.Factory
22      {
26          public static class GenericFactory
27          {
34              public static T Initialize<T>(this T item) where T : Identifier
35              {
36                  T result = item;
37                  result.Id = Guid.NewGuid();
38                  return result;
39              }
40
49              public static T Initialize<T>(this T item, XElement content,
                DataTemplate template) where T : DataContent
50              {
51                  T result = Initialize(item);
52                  result.Content = content;
53                  result.Template = template;
54                  return result;
55              }
56
65              public static T Initialize<T>(this T item, XElement schema, XElement
                stylesheet) where T : DataTemplate
66              {
67                  T result = Initialize(item);
68                  result.Contents = new HashSet<DataContent>();
69                  result.Schema = schema;
70                  result.Stylesheet = stylesheet;
71                  return result;
72              }
73
83              public static T Initialize<T>(this T item, string name, XElement content
                 = null, DataTemplate template = null) where T : BaseItem
84              {
85                  T result = Initialize(item, content, template);
86                  result.Name = name;
87                  result.Children = new HashSet<ChildItem>();
88                  result.OwnedLinks = new HashSet<BaseLink>();
89                  result.Relations = new HashSet<Relation>();
90                  result.OwnedContracts = new HashSet<Contract>();
91                  result.Properties = new HashSet<Property>();
92                  return result;
93              }
94
105             public static T Initialize<T>(this T item, string name, BaseItem parent,
                 XElement content = null, DataTemplate template = null) where T :
                ChildItem
106             {
107                 T result = Initialize(item, name, content, template);
108                 parent.AddChild(result);
109                 return result;
110             }
111
122             public static T Initialize<T>(this T item, string name, XElement content
                 = null, DataTemplate template = null, params BaseItem[] relations)
                where T : Relation
123             {
124                 T result = Initialize(item, name, content, template);
        1   2   3
```

```
125             result.Items = new HashSet<BaseItem>();
126             foreach (BaseItem relation in relations.OrderBy(i => i.Name).ToList
                ()) { relation.AddRelation(result); }
127             return result;
128         }
129
141         public static T Initialize<T>(this T item, string name, BaseItem owner,
            XElement content = null, DataTemplate template = null, params BaseItem[]
             items) where T : Property
142         {
143             T result = Initialize(item, name, content, template, items);
144             owner.AddProperty(result);
145             return result;
146         }
147
158         public static T Initialize<T>(this T item, string name, XElement content
             = null, DataTemplate template = null, params BaseLink[] contracts)
            where T : Asset
159         {
160             T result = Initialize(item, name, content, template);
161             result.Links = new HashSet<BaseLink>();
162             foreach (BaseLink contract in contracts.OrderBy(i => i.TimeStamp).
                ToList()) { result.AddOwnerLink(contract); }
163             return result;
164         }
165
177         public static T Initialize<T>(this T item, string name, ContractStatus
            status, XElement content = null, DataTemplate template = null, params
            BaseLink[] contracts) where T : BaseContract
178         {
179             T result = Initialize(item, name, content, template, contracts);
180             result.Status = status;
181             result.Transactions = new HashSet<BaseLink>();
182             return result;
183         }
184
197         public static T Initialize<T>(this T item, string name, BaseItem owner,
            ContractStatus status, XElement content = null, DataTemplate template =
            null, params BaseLink[] contracts) where T : Contract
198         {
199             T result = Initialize(item, name, status, content, template,
                contracts);
200             owner.AddOwnedContract(item);
201             return result;
202         }
203
216         public static T Initialize<T>(this T link, LinkType linkType,
            BaseContract baseContract, BaseItem owner, Asset product, XElement
            content = null, DataTemplate template = null) where T : BaseLink
217         {
218             T result = Initialize(link, content, template);
219             result.Id = Guid.NewGuid();
220             result.TimeStamp = DateTime.Now;
221             result.LinkType = linkType;
222             baseContract.AddTransaction(result);
223             owner.AddProductLink(result);
224             product.AddOwnerLink(result);
225             return result;
226         }
227
241         public static T Initialize<T>(this T link, LinkType linkType,
            BaseContract baseContract, BaseItem owner, Asset product, double amount,
             XElement content = null, DataTemplate template = null) where T :
            AmountLink
```

```
242             {
243                 T result = Initialize(link, linkType, baseContract, owner, product,
                     content, template);
244                 result.Amount = amount;
245                 return result;
246             }
247
261             public static T Initialize<T>(this T link, LinkType linkType,
                BaseContract baseContract, BaseItem owner, Asset product, int count,
                XElement content = null, DataTemplate template = null) where T :
                CountLink
262             {
263                 T result = Initialize(link, linkType, baseContract, owner, product,
                     content, template);
264                 result.Count = count;
265                 return result;
266             }
267         }
268     }
```

```
14
15      using System.ComponentModel.DataAnnotations;
16      using System.ComponentModel.DataAnnotations.Schema;
17
18      namespace WATB.Datamodel.Generic
19      {
24          [Table("AmountLinks")]
25          public class AmountLink : BaseLink
26          {
31              public override string ToString()
32              {
33                  return string.Format("{0}\tAmount: {1:0,000.00###}", base.ToString(
                    ), Amount);
34              }
39              [Required(ErrorMessage = "Need to specify amount of subject items.")]
40              public double Amount { get; set; }
41          }
42      }
```

```
14
15      using System.Collections.Generic;
16      using System.ComponentModel.DataAnnotations.Schema;
17
18      namespace WATB.Datamodel.Generic
19      {
23          [Table("Assets")]
24          public class Asset : BaseItem
25          {
30              public override string ToString()
31              {
32                  return string.Format("{0}\tAsset links: {1}\r\n", base.ToString(),
                    Links.Count);
33              }
34
39              [InverseProperty("Asset")]
40              public virtual ICollection<BaseLink> Links { get; set; }
41
46              public void AddOwnerLink(BaseLink link)
47              {
48                  Links.Add(link);
49                  link.Asset = this;
50              }
51          }
52      }
```

```
14
15      using System.Collections.Generic;
16      using System.ComponentModel.DataAnnotations;
17      using System.ComponentModel.DataAnnotations.Schema;
18
19      namespace WATB.Datamodel.Generic
20      {
24          [Table("Contracts")]
25          public class BaseContract : Asset
26          {
31              public override string ToString()
32              {
33                  return string.Format(
                    "{0}\tContract status {1} with {2} transactions\r\n", base.ToString
                    (), Status, Transactions.Count);
34              }
35
40              [Required]
41              public ContractStatus Status { get; set; }
42
47              [InverseProperty("Contract")]
48              public virtual ICollection<BaseLink> Transactions { get; set; }
49
54              public void AddTransaction(BaseLink link)
55              {
56                  Transactions.Add(link);
57                  link.Contract = this;
58              }
59          }
60      }
```

```
14
15      using System.Collections.Generic;
16      using System.ComponentModel.DataAnnotations;
17      using System.ComponentModel.DataAnnotations.Schema;
18
19      namespace WATB.Datamodel.Generic
20      {
24          [Table("Items")]
25          public class BaseItem : DataContent
26          {
31              public override string ToString()
32              {
33                  return string.Format(
                    "{0}\tName: {1}\r\n\tOwned links: {2} links\r\n\tChildren: {3}\r\n\t
                    Properties: {4}\r\n\tOwned contracts: {5}\r\n\tRelations: {6}\r\n",
                     base.ToString(), Name, OwnedLinks.Count, Children.Count,
                    Properties.Count, OwnedContracts.Count, Relations.Count);
34              }
39              [Required(ErrorMessage = "Name is required.")]
40              [MaxLength(240)]
41              public string Name { get; set; }
42
47              [InverseProperty("Master")]
48              public virtual ICollection<BaseLink> OwnedLinks { get; set; }
49
54              [InverseProperty("Parent")]
55              public virtual ICollection<ChildItem> Children { get; set; }
56
62              [InverseProperty("Owner")]
63              public virtual ICollection<Property> Properties { get; set; }
64
69              [InverseProperty("Owner")]
70              public virtual ICollection<Contract> OwnedContracts { get; set; }
71
76              [InverseProperty("Items")]
77              public virtual ICollection<Relation> Relations { get; set; }
78
83              public void AddProductLink(BaseLink link)
84              {
85                  OwnedLinks.Add(link);
86                  link.Master = this;
87              }
88
93              public void AddChild(ChildItem child)
94              {
95                  Children.Add(child);
96                  child.Parent = this;
97              }
98
103             public void AddProperty(Property property)
104             {
105                 Properties.Add(property);
106                 property.Owner = this;
107             }
108
113             public void AddOwnedContract(Contract contract)
114             {
115                 OwnedContracts.Add(contract);
116                 contract.Owner = this;
117             }
118
123             public void AddRelation(Relation relation)
124             {
125                 Relations.Add(relation);
        1   2   3
```

```
126             relation.Items.Add(this);
127         }
128
133         public void DeleteRelation(Relation relation)
134         {
135             Relations.Remove(relation);
136             relation.Items.Remove(this);
137         }
138     }
139 }
```

```
14
15      using System;
16      using System.ComponentModel.DataAnnotations;
17      using System.ComponentModel.DataAnnotations.Schema;
18
19      namespace WATB.Datamodel.Generic
20      {
24          [Table("Links")]
25          public class BaseLink : DataContent
26          {
31              public override string ToString()
32              {
33                  return string.Format(
                    "{0}\tLink {7} type {8}:\r\n\towner: {1} ({2})\r\n\tAsset: {3} ({4})
                    \r\n\tContract: {5} ({6})\r\n", base.ToString(), Master.Name,
                    Master.Id, Asset.Name, Asset.Id, Contract.Name, Contract.Id,
                    TimeStamp.ToString("yyyy-MM-dd HH:mm:ss tt zzz"), LinkType);
34              }
35
40              [Required(ErrorMessage = "Time stamp is required.")]
41              public DateTime TimeStamp { get; set; }
42
47              [InverseProperty("OwnedLinks")]
48              [Required(ErrorMessage = "Master is required.")]
49              public virtual BaseItem Master { get; set; }
50
55              [InverseProperty("Links")]
56              [Required(ErrorMessage = "Asset is required.")]
57              public virtual Asset Asset { get; set; }
58
63              [InverseProperty("Transactions")]
64              [Required(ErrorMessage = "Contract is required.")]
65              public virtual BaseContract Contract { get; set; }
66
71              [Required(ErrorMessage = "Need to specify the type of the link.")]
72              public LinkType LinkType { get; set; }
73          }
74      }
```

```
14
15      using System.ComponentModel.DataAnnotations;
16      using System.ComponentModel.DataAnnotations.Schema;
17
18      namespace WATB.Datamodel.Generic
19      {
23          [Table("Children")]
24          public class ChildItem : BaseItem
25          {
30              public override string ToString()
31              {
32                  return string.Format("{0}\tParent: {1} ({2})", base.ToString(),
                    Parent.Name, Parent.Id);
33              }
38              [InverseProperty("Children")]
39              [Required(ErrorMessage = "Parent is required.")]
40              public virtual BaseItem Parent { get; set; }
41          }
42      }
```

```
14
15      using System.ComponentModel.DataAnnotations;
16      using System.ComponentModel.DataAnnotations.Schema;
17
18      namespace WATB.Datamodel.Generic
19      {
23          [Table("OwnedContracts")]
24          public class Contract : BaseContract
25          {
30              public override string ToString()
31              {
32                  return string.Format("{0}\tOwner: {1} ({2})", base.ToString(),
                    Owner.Name, Owner.Id);
33              }
34
39              [InverseProperty("OwnedContracts")]
40              [Required(ErrorMessage = "Owner is required.")]
41              public virtual BaseItem Owner { get; set; }
42          }
43      }
```

```
14
15  namespace WATB.Datamodel.Generic
16  {
20      public enum ContractStatus
21      {
25          InProgress,
26
30          Closed,
31
35          Canceled
36      }
37  }
```

```
14
15      using System.ComponentModel.DataAnnotations;
16      using System.ComponentModel.DataAnnotations.Schema;
17
18      namespace WATB.Datamodel.Generic
19      {
23          [Table("CountLinks")]
24          public class CountLink : BaseLink
25          {
30              public override string ToString()
31              {
32                  return string.Format("{0}\tCount: {1}", base.ToString(), Count);
33              }
38              [Required(ErrorMessage = "Need to specify the number of subject items.")
                ]
39              public int Count { get; set; }
40          }
41      }
```

```
14
15      using System.ComponentModel.DataAnnotations.Schema;
16      using System.Xml.Linq;
17
18      namespace WATB.Datamodel.Generic
19      {
23          [Table("Contents")]
24          public class DataContent : Identifier
25          {
30              public override string ToString()
31              {
32                  return string.Format(
                    "{0}\tContent {1} template\r\n\tContent: {2}\r\n", base.ToString(),
                     Template == null ? "without" : "with", Content);
33              }
37              private XElement _content;
38
43              public string ContentString { get; set; }
44
49              public DataTemplate Template { get; set; }
50
55              [NotMapped]
56              public XElement Content
57              {
58                  get
59                  {
60                      if (_content == null)
61                      {
62                          _content = string.IsNullOrEmpty(ContentString) ? new
                            XElement(GetType().Name) : XElement.Parse(ContentString);
63                          _content.Changed += _content_Changed;
64                      }
65                      return _content;
66                  }
67                  set
68                  {
69                      _content = value ?? new XElement(GetType().Name);
70                      _content.Changed += _content_Changed;
71                      ContentString = _content.ToString();
72                  }
73              }
74
80              private void _content_Changed(object sender, XObjectChangeEventArgs e) {
                 ContentString = _content.ToString(); }
81
86              public bool Validate() { return Template == null || Template.Validate(
                Content); }
87          }
88      }
```

```
14
15      using System.Collections.Generic;
16      using System.ComponentModel.DataAnnotations.Schema;
17      using System.Xml.Linq;
18
19      namespace WATB.Datamodel.Generic
20      {
24          [Table("Templates")]
25          public class DataTemplate : Identifier
26          {
31              public override string ToString()
32              {
33                  return string.Format(
                    "{0}\tSchema:\r\n\t{1}\r\n\tStylesheet:\r\n\t{2}\r\n", base.
                    ToString(), Schema, Stylesheet);
34              }
35
39              private XElement _schema;
40
44              private XElement _stylesheet;
45
50              public virtual ICollection<DataContent> Contents { get; set; }
51
56              public string SchemaString { get; set; }
57
62              public string StylesheetString { get; set; }
63
68              [NotMapped]
69              public XElement Schema
70              {
71                  get
72                  {
73                      if (_schema == null)
74                      {
75                          _schema = string.IsNullOrEmpty(SchemaString) ? new
                            XElement(GetType().Name) : XElement.Parse(SchemaString);
76                          _schema.Changed += _schema_Changed;
77                      }
78                      return _schema;
79                  }
80                  set
81                  {
82                      _schema = value ?? new XElement(GetType().Name);
83                      _schema.Changed += _schema_Changed;
84                      SchemaString = _schema.ToString();
85                  }
86              }
87
92              [NotMapped]
93              public XElement Stylesheet
94              {
95                  get
96                  {
97                      if (_stylesheet == null)
98                      {
99                          _stylesheet = string.IsNullOrEmpty(StylesheetString) ?
                            new XElement(GetType().Name) : XElement.Parse(
                            StylesheetString);
100                         _stylesheet.Changed += _stylesheet_Changed;
101                     }
102                     return _stylesheet;
103                 }
104                 set
105                 {
        1   2   3   4
```

```
106                     _stylesheet = value ?? new XElement(GetType().Name);
107                     _stylesheet.Changed += _stylesheet_Changed;
108                     StylesheetString = _stylesheet.ToString();
109                 }
110             }
111
117             private void _schema_Changed(object sender, XObjectChangeEventArgs e) {
                SchemaString = _schema.ToString(); }
118
124             private void _stylesheet_Changed(object sender, XObjectChangeEventArgs e
                ) { StylesheetString = _stylesheet.ToString(); }
125
131             public bool Validate(XElement content) { return (Schema == null ||
                ValidateSchema(content)) & (Stylesheet == null || ValidateStylesheet(
                content)); }
132
139             public bool ValidateSchema(XElement content) { return true; }
140
147             public bool ValidateStylesheet(XElement content) { return true; }
148         }
149     }
```

```
14
15      using System;
16      using System.ComponentModel.DataAnnotations;
17      using System.ComponentModel.DataAnnotations.Schema;
18
19      namespace WATB.Datamodel.Generic
20      {
24          [Table("Identifiers")]
25          public class Identifier
26          {
31              public override string ToString()
32              {
33                  return string.Format("Type: {0}, ID: {1}\r\n", GetType().Name, Id);
34              }
39              [Required(ErrorMessage = "Identifier is required.")]
40              public Guid Id { get; set; }
41          }
42      }
```

```
14
15  namespace WATB.Datamodel.Generic
16  {
20      public enum LinkType
21      {
25          Link,
26
30          Send,
31
35          Receive
36      }
37  }
```

```
14
15      using System.ComponentModel.DataAnnotations.Schema;
16
17      namespace WATB.Datamodel.Generic
18      {
22          [Table("Properties")]
23          public class Property : Relation
24          {
29              public override string ToString()
30              {
31                  return string.Format("{0}\r\n\tOwner: {1} ({2})", base.ToString(),
                    Owner.Name, Owner.Id);
32              }
33
38              [InverseProperty("Properties")]
39              public BaseItem Owner { get; set; }
40          }
41      }
```

```
14
15  using System;
16  using System.Collections.Generic;
17  using System.ComponentModel.DataAnnotations.Schema;
18
19  namespace WATB.Datamodel.Generic
20  {
24      [Table("Relations")]
25      public class Relation : BaseItem
26      {
31          public override string ToString()
32          {
33              return string.Format("{0}\t{1} items", base.ToString(), Items.Count
                );
34          }
35
40          [InverseProperty("Relations")]
41          public virtual ICollection<BaseItem> Items { get; set; }
42
50          public void Add(BaseItem item)
51          {
52              if (item == this) { throw new Exception(
                "Relation cannot be added to itself."); }
53              if (Items.Contains(this)) { throw new Exception(
                "Item already exists."); }
54              Items.Add(item);
55              item.Relations.Add(this);
56          }
57
63          public void Delete(BaseItem item)
64          {
65              if (!Items.Contains(this)) { throw new Exception(
                "Item does not exists."); }
66              Items.Remove(item);
67              item.Relations.Remove(this);
68          }
69      }
70  }
```

```
1   using System;
2   using System.IO;
3   using System.Linq;
4   using Microsoft.VisualStudio.TestTools.UnitTesting;
5   using WATB.Datamodel.Context;
6   using WATB.Datamodel.Generic;
7
8   namespace WATB.Datamodel.Testing
9   {
13      [TestClass]
14      public sealed class UnitTestContract : IDisposable
15      {
19          private const string DatabaseModel = "ContractModel";
20
25          public GenericDatamodel Db { get; private set; }
26
30          public void Dispose()
31          {
32              if (Db == null) { return; }
33              Db.Dispose();
34              Db = null;
35          }
36
40          [TestInitialize]
41          public void Initialize() { Db = new GenericDatamodel(DatabaseModel); }
42
46          [TestCleanup]
47          public void WhatIsInTheDatabase()
48          {
49              using (var database = new GenericDatamodel(DatabaseModel))
50              {
51                  var idList = database.Identifiers.ToList();
52                  var path = AppDomain.CurrentDomain.BaseDirectory;
53                  using (var output = new StreamWriter(Path.Combine(path,
                    DatabaseModel + ".txt"), true))
54                  {
55                      output.WriteLine(
                        "{0}\r\nNumber of records: {1}\r\nTime: {2}\r\n{0}", new
                        string('-', 80), idList.Count, DateTime.Now.ToString(
                        "yyyy-MM-dd HH:mm:ss tt zzz"));
56                      foreach (var identifier in idList)
57                      {
58                          try { output.WriteLine(identifier); }
59                          catch (Exception ex)
60                          {
61                              output.WriteLine("\t{0} ({1})\r\n{2}",
                                identifier.GetType().Name, identifier.Id, ex.
                                Message);
62                          }
63                          output.WriteLine(new string('-', 80));
64                      }
65                      output.Close();
66                  }
67              }
68          }
69
74          [ClassInitialize]
75          public static void ClassInit(TestContext context)
76          {
77              using (var db = new GenericDatamodel(DatabaseModel))
78              {
79                  if (db.Database.Exists()) { db.Database.Delete(); }
80                  db.Database.Create();
81              }
        1   2   3
```

```
1 2 3
82          }
83
87          [TestMethod]
88          public void TestMakeContract()
89          {
90              BaseItem owner = Db.AddItem("Owner");
91              BaseItem recipient = Db.AddItem("Recipient");
92              Asset asset = Db.AddAsset("Product");
93              Db.SaveChanges();
94
95              BaseContract baseContract = Db.AddBaseContract("Contract",
                ContractStatus.InProgress);
96              Assert.IsNotNull(baseContract, "Failed to create contract.");
97              Assert.AreEqual(baseContract.Status, ContractStatus.InProgress,
                "Invalid status for contract.");
98              Assert.AreEqual(baseContract.Transactions.Count, 0,
                "Invalid number of transactions.");
99              Db.SaveChanges();
100
101             BaseLink send = Db.AddSentLink(baseContract, owner, asset);
102             Assert.IsNotNull(send, "Failed to create Send link.");
103             Assert.AreEqual(send.Contract, baseContract,
                "Send link has the wrong contract.");
104             Assert.AreEqual(send.Master, owner, "Link has the wrong owner.");
105             Assert.AreEqual(send.Asset, asset, "Link has the wrong product.");
106             Assert.AreEqual(send.LinkType, LinkType.Send,
                "Link is of the wrong type.");
107             Assert.AreEqual(baseContract.Transactions.Count, 1,
                "Invalid number of transactions.");
108             Assert.AreEqual(owner.OwnedLinks.Count, 1,
                "Owner doesn't have a product.");
109             Assert.AreEqual(asset.Links.Count, 1,
                "Product doesn't have an owner.");
110             Db.SaveChanges();
111
112             BaseLink receive = Db.AddReceivedLink(baseContract, recipient,
                asset);
113             Assert.IsNotNull(receive, "Failed to create Receive link.");
114             Assert.AreEqual(receive.Contract, baseContract,
                "Receive link has the wrong contract.");
115             Assert.AreEqual(receive.Master, recipient,
                "Link has the wrong owner.");
116             Assert.AreEqual(receive.Asset, asset, "Link has the wrong product."
                );
117             Assert.AreEqual(receive.LinkType, LinkType.Receive,
                "Link is of the wrong type.");
118             Assert.AreEqual(baseContract.Transactions.Count, 2,
                "Invalid number of transactions.");
119             Assert.AreEqual(asset.Links.Count, 2,
                "Product doesn't have a recipient.");
120             Assert.AreEqual(recipient.OwnedLinks.Count, 1,
                "Owner doesn't have a product.");
121             Db.SaveChanges();
122         }
123
127         [TestMethod]
128         public void TestMakeOwnedContract()
129         {
130             BaseItem bank = Db.AddItem("Bank");
131             BaseItem owner = Db.AddItem("Owner");
132             BaseItem recipient = Db.AddItem("Recipient");
133             Asset asset = Db.AddAsset("Product");
134             Db.SaveChanges();
135
    1 2 3
```

```
136             Contract baseContract = Db.AddContract("Contract", bank,
                ContractStatus.InProgress);
137             Assert.IsNotNull(baseContract, "Failed to create contract.");
138             Assert.AreEqual(baseContract.Owner, bank, "Failed to assign owner."
                );
139             Assert.AreEqual(baseContract.Status, ContractStatus.InProgress,
                "Invalid status for contract.");
140             Assert.AreEqual(baseContract.Transactions.Count, 0,
                "Invalid number of transactions.");
141             Db.SaveChanges();
142
143             BaseLink send = Db.AddSentLink(baseContract, owner, asset);
144             Assert.IsNotNull(send, "Failed to create Send link.");
145             Assert.AreEqual(send.Contract, baseContract,
                "Send link has the wrong contract.");
146             Assert.AreEqual(send.Master, owner, "Link has the wrong owner.");
147             Assert.AreEqual(send.Asset, asset, "Link has the wrong product.");
148             Assert.AreEqual(send.LinkType, LinkType.Send,
                "Link is of the wrong type.");
149             Assert.AreEqual(baseContract.Transactions.Count, 1,
                "Invalid number of transactions.");
150             Assert.AreEqual(owner.OwnedLinks.Count, 1,
                "Owner doesn't have a product.");
151             Assert.AreEqual(asset.Links.Count, 1,
                "Product doesn't have an owner.");
152             Db.SaveChanges();
153
154             BaseLink receive = Db.AddReceivedLink(baseContract, recipient,
                asset);
155             Assert.IsNotNull(receive, "Failed to create Receive link.");
156             Assert.AreEqual(receive.Contract, baseContract,
                "Receive link has the wrong contract.");
157             Assert.AreEqual(receive.Master, recipient,
                "Link has the wrong owner.");
158             Assert.AreEqual(receive.Asset, asset, "Link has the wrong product."
                );
159             Assert.AreEqual(receive.LinkType, LinkType.Receive,
                "Link is of the wrong type.");
160             Assert.AreEqual(baseContract.Transactions.Count, 2,
                "Invalid number of transactions.");
161             Assert.AreEqual(asset.Links.Count, 2,
                "Product doesn't have a recipient.");
162             Assert.AreEqual(recipient.OwnedLinks.Count, 1,
                "Owner doesn't have a product.");
163             Db.SaveChanges();
164         }
165     }
166 }
```

```
1   using System;
2   using System.IO;
3   using System.Linq;
4   using System.Xml.Linq;
5   using Microsoft.VisualStudio.TestTools.UnitTesting;
6   using WATB.Datamodel.Context;
7   using WATB.Datamodel.Generic;
8
9   namespace WATB.Datamodel.Testing
10  {
14      [TestClass]
15      public sealed class UnitTestItemFactory : IDisposable
16      {
20          private const string DatabaseModel = "ItemModel";
21
26          public GenericDatamodel Db { get; private set; }
27
31          public void Dispose()
32          {
33              if (Db == null) { return; }
34              Db.Dispose();
35              Db = null;
36          }
37
41          [TestInitialize]
42          public void Initialize() { Db = new GenericDatamodel(DatabaseModel); }
43
47          [TestCleanup]
48          public void WhatIsInTheDatabase()
49          {
50              using (var database = new GenericDatamodel(DatabaseModel))
51              {
52                  var idList = database.Identifiers.ToList();
53                  var path = AppDomain.CurrentDomain.BaseDirectory;
54                  using (var output = new StreamWriter(Path.Combine(path,
                    DatabaseModel + ".txt"), true))
55                  {
56                      output.WriteLine(
                        "{0}\r\nNumber of records: {1}\r\nTime: {2}\r\n{0}", new
                        string('-', 80), idList.Count, DateTime.Now.ToString(
                        "yyyy-MM-dd HH:mm:ss tt zzz"));
57                      foreach (var identifier in idList)
58                      {
59                          try { output.WriteLine(identifier); }
60                          catch (Exception ex)
61                          {
62                              output.WriteLine("\t{0} ({1})\r\n{2}",
                                identifier.GetType().Name, identifier.Id, ex.
                                Message);
63                          }
64                          output.WriteLine(new string('-', 80));
65                      }
66                      output.Close();
67                  }
68              }
69          }
70
75          [ClassInitialize]
76          public static void ClassInit(TestContext context)
77          {
78              using (var db = new GenericDatamodel(DatabaseModel))
79              {
80                  if (db.Database.Exists()) { db.Database.Delete(); }
81                  db.Database.Create();
        1   2   3   4
```

```
1 2 3 4
82              }
83          }
84
88          [TestMethod]
89          public void TestMakeTemplate()
90          {
91              DataTemplate template = Db.AddTemplate(new XElement("schema"), new
                XElement("Stylesheet"));
92              Assert.IsNotNull(template, "Failed to create template.");
93              Assert.IsNotNull(template.Schema,
                "Failed to create template schema.");
94              Assert.IsNotNull(template.Stylesheet,
                "Failed to create template stylesheet.");
95              Db.SaveChanges();
96          }
97
101         [TestMethod]
102         public void TestMakeContent()
103         {
104             DataTemplate template = Db.AddTemplate(new XElement("schema"), new
                XElement("Stylesheet"));
105             DataContent content = Db.AddContent(new XElement("Hello"), template
                );
106             Assert.IsNotNull(content, "Failed to create content.");
107             Assert.IsNotNull(content.Content, "Failed to create content data.")
                ;
108             Assert.IsNotNull(content.Template, "Failed to create template.");
109             Db.SaveChanges();
110         }
111
115         [TestMethod]
116         public void TestMakeItem()
117         {
118             BaseItem item = Db.AddItem("Item");
119             Assert.IsNotNull(item, "Failed to create item.");
120             Db.SaveChanges();
121         }
122
126         [TestMethod]
127         public void TestMakeChild()
128         {
129             BaseItem parent = Db.AddItem("Parent");
130             ChildItem child = Db.AddChild("Child", parent);
131             Assert.IsNotNull(child, "Failed to create child.");
132             Assert.AreEqual(child.Parent, parent, "Child has the wrong parent."
                );
133             Assert.AreEqual(parent.Children.Count, 1,
                "Parent doesn't have the proper amount of children.");
134             Db.SaveChanges();
135         }
136
140         [TestMethod]
141         public void TestMakeEmptyRelation()
142         {
143             BaseItem item = Db.AddItem("Item");
144             Relation relation = Db.AddRelation("Relation");
145             Assert.IsNotNull(relation, "Failed to create empty relation.");
146             relation.Add(item);
147             Assert.AreEqual(relation.Items.Count, 1,
                "Relation doesn't have the proper amount of items.");
148             Assert.AreEqual(item.Relations.Count, 1,
                "Related item doesn't have the proper amount of relations.");
149             Db.SaveChanges();
150         }
1 2
```

```
151
155         [TestMethod]
156         public void TestMakeFilledRelation()
157         {
158             BaseItem item1 = Db.AddItem("Item 1");
159             Assert.IsNotNull(item1, "Failed to create item 1 for relation.");
160             BaseItem item2 = Db.AddItem("Item");
161             Assert.IsNotNull(item2, "Failed to create item 2 for relation.");
162             Relation relation = Db.AddRelation("Relation", null, null, item1,
                item2);
163             Assert.IsNotNull(relation, "Failed to create empty relation.");
164             Assert.AreEqual(relation.Items.Count, 2,
                "Relation doesn't have the proper amount (2) of items.");
165             Assert.AreEqual(item1.Relations.Count, 1,
                "Related item 1 doesn't have the proper amount of relations.");
166             Assert.AreEqual(item2.Relations.Count, 1,
                "Related item 2 doesn't have the proper amount of relations.");
167             Db.SaveChanges();
168         }
169
173         [TestMethod]
174         public void TestDeleteRelation()
175         {
176             BaseItem item1 = Db.AddItem("Item 1");
177             BaseItem item2 = Db.AddItem("Item");
178             Relation relation = Db.AddRelation("Relation", null, null, item1,
                item2);
179             Db.SaveChanges();
180             item1.DeleteRelation(relation);
181             Assert.AreEqual(relation.Items.Count, 1,
                "Relation doesn't have the proper amount (1) of items.");
182             Assert.AreEqual(item1.Relations.Count, 0,
                "Related item 1 doesn't have the proper amount of relations.");
183             Assert.AreEqual(item2.Relations.Count, 1,
                "Related item 2 doesn't have the proper amount of relations.");
184             Db.SaveChanges();
185         }
186
190         [TestMethod]
191         public void TestMakeEmptyProperty()
192         {
193             BaseItem owner = Db.AddItem("Owner");
194             BaseItem item = Db.AddItem("Item");
195             Property property = Db.AddProperty("Property", owner);
196             Assert.IsNotNull(property, "Failed to create empty relation.");
197             Assert.AreEqual(property.Owner, owner, "Failed to assign owner.");
198             item.AddRelation(property);
199             Assert.AreEqual(owner.Properties.Count, 1,
                "Owner doesn't have the proper amount of properties.");
200             Assert.AreEqual(property.Items.Count, 1,
                "Property doesn't have the proper amount of relations.");
201             Assert.AreEqual(item.Relations.Count, 1,
                "Related item doesn't have the proper amount of relations.");
202             Db.SaveChanges();
203         }
204
208         [TestMethod]
209         public void TestMakeFilledProperty()
210         {
211             BaseItem owner = Db.AddItem("Owner");
212             BaseItem item1 = Db.AddItem("Item 1");
213             BaseItem item2 = Db.AddItem("Item");
214             Property property = Db.AddProperty("Property", owner, null, null,
                item1, item2);
```

```
215             Assert.IsNotNull(property, "Failed to create empty relation.");
216             Assert.AreEqual(property.Owner, owner, "Failed to assign owner.");
217             Assert.AreEqual(property.Items.Count, 2,
                "Relation doesn't have the proper amount (2) of items.");
218             Assert.AreEqual(item1.Relations.Count, 1,
                "Related item 1 doesn't have the proper amount of relations.");
219             Assert.AreEqual(item2.Relations.Count, 1,
                "Related item 2 doesn't have the proper amount of relations.");
220             Db.SaveChanges();
221         }
222
226         [TestMethod]
227         public void TestDeleteProperty()
228         {
229             BaseItem owner = Db.AddItem("Owner");
230             BaseItem item1 = Db.AddItem("Item 1");
231             BaseItem item2 = Db.AddItem("Item");
232             Property property = Db.AddProperty("Property", owner, null, null,
                item1, item2);
233             Db.SaveChanges();
234             item1.DeleteRelation(property);
235             Assert.AreEqual(property.Items.Count, 1,
                "Relation doesn't have the proper amount (1) of items.");
236             Assert.AreEqual(item1.Relations.Count, 0,
                "Related item 1 doesn't have the proper amount of relations.");
237             Assert.AreEqual(item2.Relations.Count, 1,
                "Related item 2 doesn't have the proper amount of relations.");
238             Db.SaveChanges();
239         }
240
244         [TestMethod]
245         public void TestMakeSubject()
246         {
247             Asset subject = Db.AddAsset("Subject");
248             Assert.IsNotNull(subject, "Failed to create subject.");
249             Db.SaveChanges();
250         }
251
255         [TestMethod]
256         public void TestMakeEmptyContract()
257         {
258             BaseContract baseContract = Db.AddBaseContract("Contract",
                ContractStatus.InProgress);
259             Assert.IsNotNull(baseContract, "Failed to create contract.");
260             Assert.AreEqual(baseContract.Status, ContractStatus.InProgress,
                "Invalid status for contract.");
261             Db.SaveChanges();
262         }
263
267         [TestMethod]
268         public void TestMakeEmptyOwnedContract()
269         {
270             BaseItem bank = Db.AddItem("Bank");
271             Contract baseContract = Db.AddContract("Contract", bank,
                ContractStatus.InProgress);
272             Assert.IsNotNull(baseContract, "Failed to create contract.");
273             Assert.AreEqual(baseContract.Owner, bank, "Failed to assign owner."
                );
274             Assert.AreEqual(baseContract.Status, ContractStatus.InProgress,
                "Invalid status for contract.");
275             Db.SaveChanges();
276         }
277
278     }
```

```
1
279     }
```

```
1   using System;
2   using System.IO;
3   using System.Linq;
4   using Microsoft.VisualStudio.TestTools.UnitTesting;
5   using WATB.Datamodel.Context;
6   using WATB.Datamodel.Generic;
7
8   namespace WATB.Datamodel.Testing
9   {
13      [TestClass]
14      public sealed class UnitTestLinkItems : IDisposable
15      {
19          private const string DatabaseModel = "LinkModel";
20
25          public GenericDatamodel Db { get; private set; }
26
30          public void Dispose()
31          {
32              if (Db == null) { return; }
33              Db.Dispose();
34              Db = null;
35          }
36
40          [TestInitialize]
41          public void Initialize() { Db = new GenericDatamodel(DatabaseModel); }
42
46          [TestCleanup]
47          public void WhatIsInTheDatabase()
48          {
49              using (var database = new GenericDatamodel(DatabaseModel))
50              {
51                  var idList = database.Identifiers.ToList();
52                  var path = AppDomain.CurrentDomain.BaseDirectory;
53                  using (var output = new StreamWriter(Path.Combine(path,
                    DatabaseModel + ".txt"), true))
54                  {
55                      output.WriteLine(
                        "{0}\r\nNumber of records: {1}\r\nTime: {2}\r\n{0}", new
                        string('-', 80), idList.Count, DateTime.Now.ToString(
                        "yyyy-MM-dd HH:mm:ss tt zzz"));
56                      foreach (var identifier in idList)
57                      {
58                          try { output.WriteLine(identifier); }
59                          catch (Exception ex)
60                          {
61                              output.WriteLine("\t{0} ({1})\r\n{2}",
                                identifier.GetType().Name, identifier.Id, ex.
                                Message);
62                          }
63                          output.WriteLine(new string('-', 80));
64                      }
65                      output.Close();
66                  }
67              }
68          }
69
74          [ClassInitialize]
75          public static void ClassInit(TestContext context)
76          {
77              using (var db = new GenericDatamodel(DatabaseModel))
78              {
79                  if (db.Database.Exists()) { db.Database.Delete(); }
80                  db.Database.Create();
81              }
        1   2   3
```

```
1 2 3
82          }
83
87          [TestMethod]
88          public void TestMakeLink()
89          {
90              BaseItem owner = Db.AddItem("Owner");
91              Asset product = Db.AddAsset("Asset");
92              BaseContract baseContract = Db.AddContract("Contract", owner,
                ContractStatus.InProgress);
93              int count = 0;
94              foreach (LinkType linkType in Enum.GetValues(typeof(LinkType)))
95              {
96                  count++;
97                  BaseLink link = Db.AddLink(linkType, baseContract, owner,
                    product);
98                  Assert.IsNotNull(link, string.Format(
                    "Failed to create link for type {0}.", linkType));
99                  Assert.AreEqual(link.Master, owner,
                    "Link has the wrong owner.");
100                 Assert.AreEqual(link.Asset, product,
                    "Link has the wrong product.");
101                 Assert.AreEqual(link.LinkType, linkType,
                    "Link is of the wrong type.");
102                 Assert.AreEqual(owner.OwnedLinks.Count, count,
                    "Owner doesn't have a product.");
103                 Assert.AreEqual(product.Links.Count, count,
                    "Product doesn't have an owner.");
104                 Db.SaveChanges();
105             }
106         }
107
111         [TestMethod]
112         public void TestMakeIntegerLink()
113         {
114             BaseItem owner = Db.AddItem("Owner, number");
115             Asset product = Db.AddAsset("Asset, number");
116             BaseContract baseContract = Db.AddContract("Contract, number",
                owner, ContractStatus.InProgress);
117             int count = 0;
118             foreach (LinkType linkType in Enum.GetValues(typeof(LinkType)))
119             {
120                 count++;
121                 BaseLink link = Db.AddLink(linkType, baseContract, owner,
                    product, 10);
122                 Assert.IsNotNull(link, string.Format(
                    "Failed to create count link for type {0}.", linkType));
123                 Assert.IsInstanceOfType(link, typeof(CountLink),
                    "Link is of the wrong type.");
124                 Assert.AreEqual(link.Master, owner,
                    "Link has the wrong owner.");
125                 Assert.AreEqual(link.Asset, product,
                    "Link has the wrong product.");
126                 Assert.AreEqual(link.LinkType, linkType,
                    "Link is of the wrong type.");
127                 Assert.AreEqual(owner.OwnedLinks.Count, count,
                    "Owner doesn't have a product.");
128                 Assert.AreEqual(product.Links.Count, count,
                    "Product doesn't have an owner.");
129                 Db.SaveChanges();
130             }
131         }
132
136         [TestMethod]
137         public void TestMakeFloatLink()
    1 2
```

```
138             {
139                 BaseItem owner = Db.AddItem("Owner, amount");
140                 Asset product = Db.AddAsset("Asset, amount");
141                 BaseContract baseContract = Db.AddContract("Contract, amount",
                    owner, ContractStatus.InProgress);
142                 int count = 0;
143                 foreach (LinkType linkType in Enum.GetValues(typeof(LinkType)))
144                 {
145                     BaseLink link = Db.AddLink(linkType, baseContract, owner,
                        product, Math.PI);
146                     count++;
147                     Assert.IsNotNull(link, string.Format(
                        "Failed to create amount link for type {0}.", linkType));
148                     Assert.IsInstanceOfType(link, typeof(AmountLink),
                        "Link is of the wrong type.");
149                     Assert.AreEqual(link.Master, owner,
                        "Link has the wrong owner.");
150                     Assert.AreEqual(link.Asset, product,
                        "Link has the wrong product.");
151                     Assert.AreEqual(link.LinkType, linkType,
                        "Link is of the wrong type.");
152                     Assert.AreEqual(owner.OwnedLinks.Count, count,
                        "Owner doesn't have a product.");
153                     Assert.AreEqual(product.Links.Count, count,
                        "Product doesn't have an owner.");
154                     Db.SaveChanges();
155                 }
156             }
157         }
158     }
```

T

U

V

W

X