LINQ to XML and Objects

XML file

```
<?xml version="1.0" encoding="utf-8"?>
<Customers>
 <Customer ID="1">
  <Forename>Joe</Forename>
  <Surname>Stevens</Surname>
  <DOB>31/01/1983</DOB>
  <Location>Sydney</Location>
 </Customer>
 <Customer ID="2">
  <Forename>Tom</Forename>
  <Surname>Male</Surname>
  <DOB>02/02/1977</DOB>
  <Location>Brisbane</Location>
 </Customer>
 <Customer ID="3">
  <Forename>Emily </Forename>
  <Surname>Stevens</Surname>
  <DOB>14/01/1988</DOB>
  <Location>Sydney</Location>
 </Customer>
 <Customer ID="4">
  <Forename>Lee</Forename>
  <Surname>Phipps</Surname>
  <DOB>05/12/1982</DOB>
  <Location>Melbourne</Location>
 </Customer>
 <Customer ID="5">
  <Forename>Saul</Forename>
  <Surname>Stevens</Surname>
  <DOB>02/08/1984</DOB>
  <Location>Perth</Location>
 </Customer>
</Customers>
```

A Customer class to represent each customer:

```
public class Customer
{
    public int ID { get; set; }
    public string Forename { get; set; }
    public string Surname { get; set; }
    public string DOB { get; set; }
    public string Location { get; set; }
}
```

Select a single customer based on their ID. The following method shows how to load the XML file and query it to find the customer we want, and return the data as a single Customer object:

```
public static Customer GetCustomer(int customerID)
{
    XDocument data = XDocument.Load(/Data/Customers.xml");

    return (from c in data.Descendants("Customer")
            where c.Attribute("ID").Value.Equals(customerID.ToString())
            select new Customer()
            {
                ID = Convert.ToInt32(c.Attribute("ID").Value),
                Forename = c.Element("Forename").Value,
                Surname = c.Element("Surname").Value,
                DOB = c.Element("DOB").Value,
                Location = c.Element("Location").Value

            }).FirstOrDefault();
}
```

To get a list of all customers is very similar to getting a single customer, although the method will return a generic list of Customer objects:

```
public static List<Customer> GetCustomers()

{
    XDocument data = XDocument.Load(/Data/Customers.xml");

    return (from c in data.Descendants("Customer")
            orderby c.Attribute("Surname")
            select new Customer()
            {
                ID = Convert.ToInt32(c.Attribute("ID").Value),
                Forename = c.Element("Forename").Value,
                Surname = c.Element("Surname").Value,
                DOB = c.Element("DOB").Value,
                Location = c.Element("Location").Value

            }).ToList();
}
```

A single method called Save which is used for both inserting and updating. It accepts a Customer object and performs an insert or update depending on the ID value of that object:

```
public static void Save(Customer customer)
{
    XDocument data = XDocument.Load(/Data/Customers.xml");

    if (customer.ID > 0)
    {
        XElement customerElement = data.Descendants("Customer").Where(c =>
c.Attribute("ID").Value.Equals(customer.ID.ToString())).FirstOrDefault();
        if (customerElement != null)
        {
            customerElement.SetElementValue("Forename", customer.Forename);
            customerElement.SetElementValue("Surname", customer.Surname);
            customerElement.SetElementValue("DOB", customer.DOB);
            customerElement.SetElementValue("Location", customer.Location);

            data.Save("/Data/Customers.xml");
        }
    }
    else
    {
        XElement newCustomer = new XElement (   "Customer",
                                                new XElement("Forename",
customer.Forename),
                                                new XElement("Surname",
customer.Surname),
                                                new XElement("DOB",
customer.DOB),
                                                new XElement("Location",
customer.Location)
                                            );

        newCustomer.SetAttributeValue("ID", GetNextAvailableID());

        data.Element("Customers").Add(newCustomer);
        data.Save("~/Data/Customers.xml");
    }
}
```

The GetNextAvailableID method simply finds the highest ID used in the XML document and adds one to it:

```
private static int GetNextAvailableID()
{
    XDocument data = XDocument.Load("~/Data/Customers.xml");

    return Convert.ToInt32(
             (from c  in data.Descendants("Customer")
              orderby Convert.ToInt32(c.Attribute("ID").Value) descending
              select c.Attribute("ID").Value).FirstOrDefault()
            ) + 1;
}
```

Deleting a customer from the XML file:

```
{
    XDocument data = XDocument.Load("/Data/Customers.xml");

    XElement customerElement = data.Descendants("Customer").Where(c =>
c.Attribute("ID").Value.Equals(customer.ID.ToString())).FirstOrDefault();
    if (customerElement != null)
    {
        customerElement.Remove();
        data.Save("~/Data/Customers.xml");
    }
}
```