Creating your own packet types

In this small example we want to create our own XMPP extension to send weather information over the XMPP network. The easiest way to do that is to embed the data with a message stanza. Our raw XML message with embedded weather information will look like that:

```
<message xmlns="jabber:client" to="romeo@montage.net">
      <weather xmlns="agsoftware:weather">
            <humidity>90</humidity>
            <temperature>57</temperature>
      </weather>
</message>
```

We created a new namespace for our own protocol and the 3 new elements weather, humidity and temperature.

First we create a new class weather.cs for our custom XML-Element and derive from agsXMPP.Xml.Dom.Element

```
using System;
using agsXMPP.Xml.Dom;

namespace MiniClient
{
    public class Weather : Element
    {
        public Weather()
        {
            this.TagName = "weather";
            this.Namespace = "agsoftware:weather";
        }

        public Weather(int humidity, int temperature) : this()
        {
            this.Humidity       = humidity;
            this.Temperature    = temperature;
        }

        public int Humidity
        {
            get { return GetTagInt("humidity"); }
            set { SetTag("humidity", value.ToString()); }
        }

        public int Temperature
        {
            get { return GetTagInt("temperature"); }
            set { SetTag("temperature", value.ToString()); }
        }

    }
}
```

We have to register the new class in the ElementFactory. Without registering the class the XML-Parser is not able to build or weather objects while parsing the XML-Stream. We register the class with the following line of code:

```
agsXMPP.Factory.ElementFactory.AddElementType("weather",
"agsoftware:weather", typeof(Weather));
```

You should register your own Elements before doing anything else with agsXMPP.

Now we can build our weather message and send it:

```
Weather weather = new Weather(90, 57);

Jid to = new Jid("romeo@montage.net");
Message msg = new Message();
msg.To = to;
// Add our weather Element
msg.AddChild(weather);
// Send the message
XmppCon.Send(msg);
```

Another application which receives this message can access our custom data like in this OnMessage handler:

```
private void XmppCon_OnMessage(object sender, Message msg)
{
      if (msg.HasTag(typeof(Weather)))
      {
            Weather weather = msg.SelectSingleElement(typeof(Weather)) as
      Weather;
            Console.WriteLine(weather.Temperature.ToString());
            Console.WriteLine(weather.Humidity.ToString());
      }
}
```

Now we created our first own packet and included with a message stanza. In the next step we create a small weather service which is based on Iqs (info queries).

raw xml protocol of our service:

Romeo requests the weather information for his city with the zip code ‘74080’ from the weather service:

```
<iq from=’romeo@montagne.net to=’weather.mortagne.net’ type=’get’
id=’agsXMPP_1’>
      <query xmlns=’agsoftware:weather’>
            <zip>74080</zip>
      </query>
</iq>
```

the weather service looks up the weather data for the requested zip code and sends the result back to Romeo

```
<iq to='romeo@montagne.net' from='weather.mortagne.net' type='result'
id='agsXMPP_1'>
      <query xmlns='agsoftware:weather'>
            <humidity>90</humidity >
            <temperature>57</temperature>
            <zip>74080</zip>
      </query>
</iq>
```

we create 2 classes, Weather.cs and WeatherIq.cs. The Weather class represents the <query/> object, and the WeatherIq class represents the whole Iq.

source code Weather.cs:

```
using System;

using agsXMPP.Xml;
using agsXMPP.Xml.Dom;

public class Weather : Element
{
    public Weather()
    {
        this.TagName = "query";
        this.Namespace = "agsoftware:weather";
    }

    public int Humidity
    {
        get { return GetTagInt("humidity"); }
        set { SetTag("humidity", value.ToString()); }
    }

    public int Temperature
    {
        get { return GetTagInt("temperature"); }
        set { SetTag("temperature", value.ToString()); }
    }

    public int Zip
    {
        get { return GetTagInt("zip"); }
        set { SetTag("zip", value.ToString()); }
    }
}
```

source code WeatherIq.cs:

```
using System;

using agsXMPP;
using agsXMPP.protocol.client;

public class WeatherIq : IQ
{
    private Weather m_Weather = new Weather();

    public WeatherIq()
    {
        base.Query = m_Weather;
        this.GenerateId();
    }

    public WeatherIq(IqType type)
        : this()
    {
        this.Type = type;
    }

    public WeatherIq(IqType type, Jid to)
        : this(type)
    {
        this.To = to;
    }

    public WeatherIq(IqType type, Jid to, Jid from)
        : this(type, to)
    {
        this.From = from;
    }

    public new Weather Query
    {
        get
        {
            return m_Weather;
        }
    }
}
```

of course we have to register the Weather object in the factory with the following line of code:

```
agsXMPP.Factory.ElementFactory.AddElementType("weather",
"agsoftware:weather", typeof(Weather));
```

Romeo's client builds the requesting Iq packet using the WeatherIq object:

```
WeatherIq wIq = new WeatherIq(IqType.get);
wIq.To = new Jid("weather.mortagne.net");
wIq.From = new Jid("romeo@montagne.net");
wIq.Query.zip = 74080;
// Send the message
XmppCon.Send(wIq);
```

The weather service receives the request in the Iq handler and sends the response to Romeo. We can take the incoming element from Romeo and modify it for the response.

```
private void XmppCon_OnIq(object sender, agsXMPP.Xml.Dom.Node e)
{
      IQ iq = e as IQ;
      Element query = iq.Query;

      if (query != null)
      {
            if (query.GetType() == typeof(Weather))
            {
                  // its a Weather IQ
                  Weather weather = query as Weather;
                  if (iq.Type == IqType.get)
                  {
                        Console.WriteLine(weather.Zip.ToString());
                        // read the zip code and lookup the weather
                        // data for this zip code
                        // . . . .
                        iq.SwitchDirection();
                        iq.Type = IqType.result;
                        weather.Humidity        = 90;
                        weather.Temperature     = 57;
                        // Send the result back to romeo
                        XmppCon.Send(iq);
                  }
            }
      }
}
```

Romeo will recieve the response and can access the weather data with the properties of our weather object.

We hope this tutorial helps you to build your own packets, and will give you some ideas how easy it is to build new cool services based on XMPP with the powerful agsXMPP library.