

RSS Feed mit dem DataSet

Autor: sclearscreen

IDE: Visual Studio 2003 Standard

Betriebssystem: WindowsXP

Vorwort:

Ganz kurz gesagt wer sich meinen vorherigen Beitrag angeschaut hat? Der wird nicht überlesen haben das ein Bug sein Unwesen in der kleinen enstanden Version sein Unwesen treibt.

Es ist kein schwerwiegender Fehler der den ganzen Desktop-PC zum abstürzen bringt. Dieser Bug veranlasst auch nicht das Programm sich ins System verabschiedet. Nein das Programm parst beim Umgang mit RSS Feed einer bestimmten Version nicht alle Daten aus. Sprich das Parsen des XML funktioniert nicht richtig. Bei dem Algorithmus zum parsen des XML hatte ich voll auf Eigenbau gesetzt was nicht ganz von Erfolg gekrönt war. Also auf ans Werk.

Inhalt:

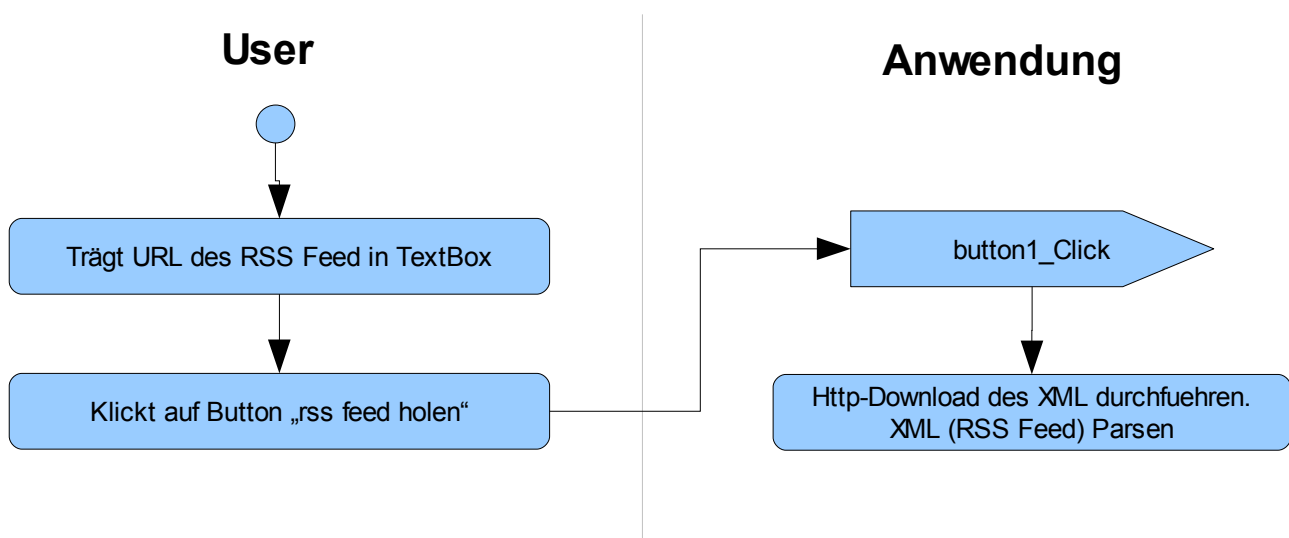
- 1 Refactoring des Programm
- 1.1 [Wo muss das Refactoring angesetzt werden?](#)
- 1.2 [Was hat das Refactoring am Code verändert?](#)
- 2 [Was lernt man daraus?](#)
- 3 [Zum Testen gebe ich dem Der/Die möchte die Lösung nochmal auf den Weg](#)
- 3.1 [Die Methode ohne Verwendung eines Proxyserver](#)
- 3 [Die Methode mit Verwendung eines Proxyserver](#)
- 4 [Haftungsausschluss fuer etwaige Schäden an Hardware/Software des User](#)

1. Refactoring des Programm

1.1 Wo muss das Refactoring angesetzt werden?

Ich werde vorher mit Hilfe von UML den Weg der Daten bis zum Bug deutlich machen und die fehlerhafte Implementierung dieser Stelle aufzeigen. Sollte manches an der UML-Symbolik nicht ganz so stimmen bitte ich das zu verschmerzen. Ich benutzte OpenOffice was nicht alle alle Standardformen Rechtecke, Verbindungspfeile bereitstellt. Somit ist es mir nicht möglich spezielle Besonderheiten bei UML zu berücksichtigen. Beispielsweise kann man durch Verbindungspfeile kein asynchrones Verhalten bei Prozessaufrufen darstellen, wo man eine Verbindungspfeil mit einer halben Pfeilspitze zur Darstellung braucht. So genug abgeschweift zur Sache.

Wir beginnen mit einem *Aktivitätsdiagramm* was die Interaktion des User mit seinem Reader kurz nach dem Start der Anwendung zeigt.



Ich hoffe jetzt fällt dem aufmerksamen Leser etwas auf was uns die UML-Symbolik als Signal verkauft/darstellt, ist die Methode die als Callback-Funktion fuer den Button genutzt wird. Und eben diese Callback-Funktion stoest ihrerseits in Ihrem Rumpf den Code zum Dowload und Parsen des Feed an. Genau dort liegt der Hase im Pfeffer. In diesem Code interagieren/kollaborieren 4 Klassenobjekte miteinander um den das XML-Script des RSS Feed per Http zu holen. Eines der Klassenobjekte ist eine Instanz des Types XmlTextReader. Der Algorithmus des Parsen bestand nun darin dieses Objekt mit den Kernablaufstrukturen, die jeder von Euch kennt, sinnvoll zu benutzen. Sinn und Zweck dieser Massnahme ist an alle item-Tags zu kommen die wiederum unserer Informationen enthalten die der Nutzer aus dem Feed lesen möchte.

Nun kann man also folgenden Block

Http-Download des XML durchfuehren.
XML (RSS Feed) Parsen.

der ganz abstract gesehen das innerer der Methode
([diese Methode siehe meinem Beispielcode zum downloaden](#))

```
private void button1_Click(object sender, System.EventArgs e)
{
    .
    .
    .
}
```

darstellt wobei man im Vorfeld sagen kann das an der Beschaffung des XML-Scriptes per HTTP innerhalb der Methode nichts einzuwenden ist. Auch die Tatsache das dort die Instanz des Types *XmlTextReader* den Stream bekommt hat nichts mit dem Bug zu tun. Wir sind also bis zu folgendem Punkt gekommen wo bisher alles reibungslos läuft.

```
XmlTextReader xmltextReader = new XmlTextReader(this.rssFeed.GetResponse().GetResponseStream());
```

Nach dieser Zuweisung beginnt der Eigenbaualgorithmus zum Parsen des XML der sich als zu unfehlbar gegenüber RSS Feed der Version 1.0 erweist. 2 der der 3 notwendigen Tags werden nicht ausgeparst, soweit läuft das Programm ohne Endlosschleifen durch. Ich hatte mir an der Stelle um den Fehler ohne zwangsläufig auf eine Netzverbindung angewiesen zu sein. Einen Feed der Version 1.0 auf meine Platte geladen da die Klasse *XmlTextReader* auch aus Dateien lesen kann habe ich die Instanzierung auf diese XML-Datei umgestellt. Somit konnte ich problemloser Schrittweise debuggen. Wobei man feststellt das der Algorithmus dann eben bei jedem „item-Tag“ nur einen der 3 Tags ausparst um dann zum naechsten „item-Tag“ zu springen.

Ich gebe zu an dem Punkt hatte mir dann die Muse gefehlt um den Algorithmus zu verbessern. Die Gefahr einer Endlosschleifen innerhalb der while-Schleife erachtet ich zu gross. Eine einfacherer Lösung auf Basis von vorhanden Komponenten ist immer besser als das Rad weiter neu zu erfinden.

Da ich wusste das die Klasse *DataSet* mit XML umgehen kann gerade im Bezug auf Datenbanken. Versuchte ich 1 und 1 zusammenzuzählen. RSS Feed aufgrund ihrer interen Struktur datenzentriert also ideal zur arbeit mit Datenbanken ausgelegt. Hinzu kommt das DataSet kann XML Laden als auch Speichern, beim Laden kann es seine Daten aus einem *Stream*-Objekt bekommen als auch von einem *XmlTextReader*-Objekt.

Also wenn sich diese Gedanken nicht gut anhören, es ergab sich eine funktionsfähige Lösung innerhal von 10-15 Minuten. Das Hilfesystem und die MSDN haben sich dabei wiederum bewährt.

1.2 Was hat das Refactoring am Code verändert?

Folgenden Code wurde somit aus dem Projekt durch auskommentieren verbannt:

```
while(xmltextReader.Read() == true)
{
    if(xmltextReader.Name == "item") /
    {
        while(xmltextReader.Read()) // weiterlesen im Element <ITEM>
        {
            switch(xmltextReader.Name) // lesen bis wir ....
            {
                case "title":
                {
                    if(titlereaded != true)
                    {
                        while(xmltextReader.Read() == true &&
                            xmltextReader.NodeType != XmlNodeType.Text);

                        this.textBox2.Text = this.textBox2.Text + "TITLE: " +
                            xmltextReader.Value + "\r\n";
                        titlereaded = true; // merken Uns das Titel gefunden ist
                    }
                    break;
                }

                case "link":
                {
                    if(linkreaded != true) // doppeltes Lesen verhindern
                    {
                        while(xmltextReader.Read() == true &&
                            xmltextReader.NodeType != XmlNodeType.Text);

                        this.textBox2.Text = this.textBox2.Text + "LINK: " +
                            xmltextReader.Value + "\r\n";
                        linkreaded = true; // merken Uns das Link gefunden ist
                    }
                    break;
                }

                case "description": // und auch das 3. Kernelement Description wurde gefunden
                {
                    if(descriptionreaded != true) // doppeltes Lesen der Description verhindern
                    {
                        while(xmltextReader.Read() == true &&
                            xmltextReader.NodeType != XmlNodeType.Text);

                        this.textBox2.Text = this.textBox2.Text + "DESCRIPTION: " +
                            xmltextReader.Value + "\r\n";

                        descriptionreaded = true; // merken uns das Description ist
                    }
                    break;
                }

                default: break; // hier Sind es optionale Tags des <ITEM> - Elementes
            }

            // Wir prüfen jetzt ab ob Wir alle 3 Kernelemente des <ITEM> - Elementes gefunden haben
            if(titlereaded == true && linkreaded == true && descriptionreaded == true) //
                break; // alles nötige für diesen einen News - Feed ist
            // ausgelesen Wir springen aus der Schleife um weitere <ITEM> - Element auslesen zukönnen
        }
        titlereaded = false;
        linkreaded = false;
        descriptionreaded = false;
        this.textBox2.Text = this.textBox2.Text + "-----\r\n";
    }
}
```

dies wird ersetzt gegen folgende kurze Sequenz ich war jedenfalls froh das dieses Geschoss vom Tisch war, den der neue Code liest sich auch viel besser. Und man bekommt auch einen Wink mit dem Zaunpfahl was Wiederverwendung von getestet Komponenten ausmacht. Und das bekommt man so ganz gratis mit dem Framework dazu. Ich will niemanden auf die Folter spannen hier

also das Code-Listning:

```
DataSet ds = new DataSet("RSS-Feed: " + this.textBox1.Text);
System.IO.Stream stream = this.rssFeed.GetResponse().GetResponseStream();
XmlTextReader xmltextReader = new XmlTextReader(stream);
ds.ReadXml(xmltextReader);
DataTable dt = ds.Tables["item"];
```

Das sind 5 Zeilen Code gegenueber ich weiss nicht wieviel Zeilen diese Zeilen ersetzt die While-Schleife mit den ganzen eingeschachtelten Ablaufstrukturen.

Man brauch sich jetzt wirklich nur ein DataGrid auf sein Formular ziehen und weist desen Eigenschaft *DataSource* eine der Variablen *ds* oder *dt* zu funktioniert beides.

Im Verlauf der naechsten Sachen die man mit dem *DataGrid* anstellen kann erinnert man sich an die Sache mit dem *HitTestInfo*. Da bekommt man angeklickte bzw aktivierte Zellen im DataGrid zurueck. Somit kommt man an den Url ran der in der Spalte mit dem Namen „link“ im DataGrid auftauchen wird, nachdem man sihe folgenden Code:

```
datagrid1.DataSource = dt;
```

gemacht hat. Liest man den Inhalt ueber *HitTestInfo* aus kann man diese Information einem jeden Browser zufuehren der sich per Interoperabilität ansteuern lässt. Das ist die ganze Kunst.

2. Was lernt man daraus?

- a) Es ist immer gut und praktikabel sich mit der Klassenbibliothek des Framework auseinanderzusetzen. Es bringt bessere Lösungen.
- b) Man baut ausgiebig auf bereits getesten Code auf, was Wiederverwendbarkeit und Wartbarkeit fördert.
- c) durch Verwendung bestehender getesteter Klassen wird Code sicherer unter Umständen kürzer als auch lesbarer
- d) Die genannten Dinge bringen nur was wenn man Kenntnisse ueber das jeweilige Framework hat.

Man ist im Bezug auf die 4 Punkte immer gut beraten die MSDN, Foren und das Internet zu benutzen.

3. Zum Testen gebe ich dem Der/Die möchte die Lösung nochmal auf den Weg

folgende Methode kann man in sein Projekt kopieren, einfach in die gewünschte Klasse als Methode reinkopieren. Existiert in 2 Varianten nicht ohne Grund!!!

3.1 Die Methode ohne Verwendung eines Proxyserver

```
// Achtung Methode kann je nach Netzwerk wenn nen Proxy zwischen euch sitzt ne Exception
// werfen
private DataTable GetRssFeedItems(string Url)
{
    // Dies Teil transferiert die Daten des RSS Feed zu unseren Rechner auch hier
    // könnte ne Exception auftreten aber die Meldung bekommen Wir auch in die Statuszeile
    System.Net.HttpWebRequest rssFeed = (System.Net.HttpWebRequest)
        (WebRequest.Create(Url));

    // Der XmlTextReader bekommt einen Netzwerkstrom direkt vom HttpWebRequest -
    // Objekt
    // dies Teil könnte ne Exception werfen das sehen Wir dann aber in der Statuszeile
    DataSet ds = new DataSet("RSS-Feed: " + Url);
    System.IO.Stream stream = rssFeed.GetResponse().GetResponseStream();
    System.Xml.XmlTextReader xmltextReader = new System.Xml.XmlTextReader(stream);
    ds.ReadXml(xmltextReader);
    return (ds.Tables["item"] as DataTable);
}
```

3.2 Die Methode mit Verwendung eines Proxyserver

```
private DataTable GetRssFeedItems(string Url,
    string proxyAdresse,
    string proxyPort,
    string username,
    string passwort)
{
    // Dies Teil transferiert die Daten des RSS Feed zu unseren Rechner auch hier
    // könnte ne Exception auftreten aber die Meldung bekommen Wir auch in die Statuszeile
    System.Net.HttpWebRequest rssFeed = (System.Net.HttpWebRequest)
        (WebRequest.Create(Url));

    // ToDo: Wenn es hier ne Exception gibt, kann es daran liegen das ein Proxy zwischen uns
    // und dem Internet sitzt! Wir probieren es daher mit der Proxyauthentifizierung.
    // Man nehme diese Proxydaten aus dem Lokalen Browser
    rssFeed.Proxy = new System.Net.WebProxy(proxyAdresse + ":" + proxyPort);
    rssFeed.Proxy.Credentials = new System.Net.NetworkCredential(username, passwort);

    // Der XmlTextReader bekommt einen Netzwerkstrom direkt vom HttpWebRequest -
    // Objekt
    // dies Teil könnte ne Exception werfen das sehen Wir dann aber in der Statuszeile
    DataSet ds = new DataSet("RSS-Feed: " + Url);
    System.IO.Stream stream = rssFeed.GetResponse().GetResponseStream();
    System.Xml.XmlTextReader xmltextReader = new System.Xml.XmlTextReader(stream);
    ds.ReadXml(xmltextReader);
    return (ds.Tables["item"] as DataTable);
}
```

3.2 Verwenden kann die 2 Methoden dann beispielsweise so

```
try // Versuch den Stream normal zu holen
{
    this.dataGrid1.DataSource = GetRssFeedItems("http://www.rss-verzeichnis.de/updates.xml");
}
catch(System.Exception ex) // Exception kommt wenn ein Proxyserver dazwischen ist
{
    MessageBox.Show(ex.Message);

    try // Versuch den Stream zu holen trotz Proxy
    {
        this.dataGrid1.DataSource = GetRssFeedItems("http://www.rss-verzeichnis.de/updates.xml",
            "100.100.100.100", // P.S.: fiktive ProxyIp
            "8080",
            "AlfonzDerUser",
            "AfonzesPasswort");
    }
    catch(System.Exception ex)
    {
        // Punkt an dem es keinen Sinn macht die Anwendung weiter fortzusetzen
        // Stream konnte mit normaler Internetverbindung nicht geholt werden
        // Stream konnte auch nicht mit dem zwischengeschalteten Proxy beschafft werden

        MessageBox.Show(ex.Message);

        // Gruende:
        // - falscher Proxy (Proxydaten, Proxyserver inaktiv)
        // - Webserver mit RSS Feed inaktiv
        // - Internetverbindung aus verschiedene Gruenden inaktiv
    }
}
}
```

Wie man sieht wenn man die Methoden benutzt ist es ratsam try/catch zu verwenden ich habs ja als Kommentar in der einen Methode angekündigt

4 Haftungsausschluss fuer etwaige Schäden an Hardware/Software des User

Man kann ja nie wissen! Ich will also fast hier am Ende nochmal darauf hinweisen, das Jeder/Jede der Inhalte aus meinen Beiträgen nutzt, dies auf eigene Gefahr tut!

Ich übernehme keinerlei Haftung fuer Schäden die durch den Download, Nutzung oder Änderung an dem aufgeführten Quellcode aufgetreten sind.

Ich teile gerne Erfahrung aber man muss sich absichern gerade heute. So das wäre auch geklärt denke ich. Bleibt nur noch das Nachwort oder der Nachtrag.

Nachtrag

So damit kann ein Interessierter seine Spieltrieb freien Lauf lassen die Sache mit dem Mozillabrowser habe ich bewusst aussen vor gelassen. Einen Browser noch reinzupacken sollte kein Problem darstellen gibt Beispiele zuhauf im Netz. Ausserdem nur durch rumspielen und rumexperimentieren erlangt man Wissen. Denkt beim *DataGrid* an die Sache mit dem *HitTestInfo* ueber kurz oder lang sollte jemand fuer den das neu ist mit dem Hinweis zu Lösung gelangen.

Da kommt man dann auch uebers *DataGrid* an den Urls ran die im Feed stehen. Hier brauch man den Url dann nur noch mit der *Navigate*-Methode des Browserobjekt in Einklang bringen schon hat man den RSS-Reader fertig. Daran laesst sich noch viel machen.

Verwaltung von Urls auf die Feeds etc.