

# Serving Up RSS using Delphi 8 (part 2)

by Craig Murphy

In my last instalment in this series of articles about RSS, Delphi 8 and blogging, I described how I have set about trying to make some pages on my website more dynamic via the use of some simple PHP scripts and RSS files. Part 1 was a little disjointed and was really setting the scene and direction for this and future articles.

Over the course of this article I will be using Delphi 8 (with update 2 applied). I will be describing how I used some of the ideas from part 1 to automate the creation of my Developers Group downloads page at <http://www.craigmurphy.com/bug>. A rather cool side-effect of this will be the creation of an RSS feed for my Developers Group downloads.

## Where are we going?

Since the publication of part 1, some of you may have noticed the presence of a little orange icon on my website. If you point your browser to <http://www.craigmurphy.com/bug/bug.xml> you will find an RSS 2.0 feed that contains references to all of my BUG/DDG/DG source code and presentations. Believe it or not, I actually use that very same feed to build the content at <http://www.craigmurphy.com/bug>. This means that when I re-publish bug.xml those users who have syndicated my DG content receive an update when their RSS aggregator polls my site. However, what's a great timer-saver for me is the fact that I do not have to touch the HTML page at <http://www.craigmurphy.com/bug>.

So, over the course of this article I will be describing how I create bug.xml. In the next article in this series, I will demonstrate the PHP code that is required to create HTML from bug.xml – it is this HTML that is rendered in your browser when you visit: <http://www.craigmurphy.com/bug>.

## First Things First

We know that RSS feeds are nothing more than a specific XML format. It can assumed that we will need a means of loading and saving the RSS feed – it's an XML file, so we can make use of .NET's XmlDocument object found in the System.XML namespace.

So, given the following variable declaration:

```
uses System.Xml;  
...  
var xmlDoc : XmlDocument;
```

We could write:

```
xmlDoc := XmlDocument.Create;  
xmlDoc.Load(FileName);
```

Assuming that FileName referenced a valid XML document (or an RSS feed), xmlDoc will provide us with access to the XML document via a Document Object Model (DOM). If you are unsure about the DOM, visit <http://www.craigmurphy.com/bug> and look for "Using OpenXML and the XDOME".

More usefully however, given an XmlDocument, we have can use the SelectSingleNode method with an XPath expression to gain granular access to specific nodes with XmlDocument.

I've chosen to wrap XML node selection up and have created a method, \_\_GetNode, to help me out.

\_\_GetNode takes an XML Document and an XPath expression. It will either return the node's textual value or an empty string if the node doesn't exist. It's basic, but let's get things working before we set about improving them.

Here's how we might make use of \_\_GetNode:

```
Title := __GetNode(xmlDoc, 'rss/channel/title');
```

\_\_GetNode's implementation simply calls SelectSingleNode:

# RAIZE SOFTWARE

## CodeSite

When your code isn't doing what you expect, find out why with CodeSite



- New for CodeSite 3...**
- CodeSite Projects
  - Automatic Message Filtering
  - Automatic Views
  - XML Support
  - .NET Support
  - Custom Views
  - Enhanced Thread Support
  - Plus much more...

CodeSite helps developers locate problems in their code by enabling them to send detailed information from within their application code to a specialized receiver.

### Send Any Information

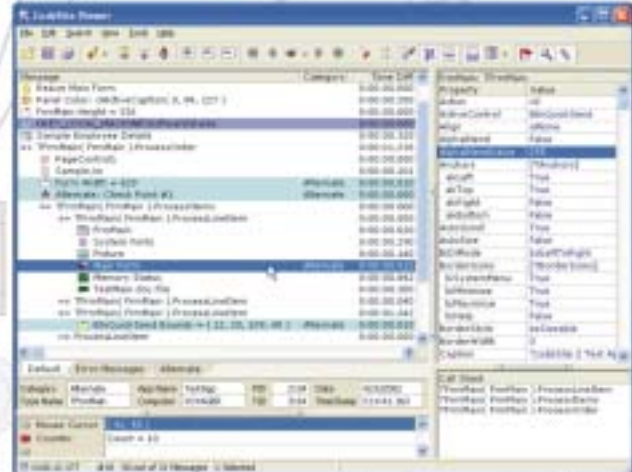
CodeSite supports sending snapshots of all kinds of data, including all standard types, objects, string lists, variants, bitmaps, files, memory statistics, registry entries, custom data, and much more.

### Receive the Messages

CodeSite messages can be sent to a variety of destinations including the CodeSite Viewer, a log file, a remote computer, and even a web server.

### Analyze the Messages

The CodeSite Viewer (screen shot) provides a powerful and highly-flexible environment for analyzing CodeSite messages.



## Raize DropMaster

OLE Drag-and-Drop made easy

### Simply Powerful

DropMaster encapsulates the complexity of OLE inter-application drag-and-drop in 4 easy-to-use components.



### Drag Sources

Easily support dragging text, images, or custom data formats to other applications.

### Drop Targets

Accept plain text, rich text, file lists, URL links, images, and custom data formats from other applications that supports OLE drag and drop.

### Customize the Process

Special events are provided so that the drag and drop process can be customized.

### Real World Examples

DropMaster comes with more than 30 example projects that demonstrate the extreme power and flexibility of these components.

### Examples Include:

- Accept messages dragged from Outlook, Outlook Express, Netscape, and Eudora.
- Dragging URLs from an application to a browser or the desktop.
- Dragging multiple custom formats
- Dragging and creating multiple files
- Accepting OLE object links
- Dragging JPEG images
- Dragging custom content to Explorer folders or the desktop

## Raize Components

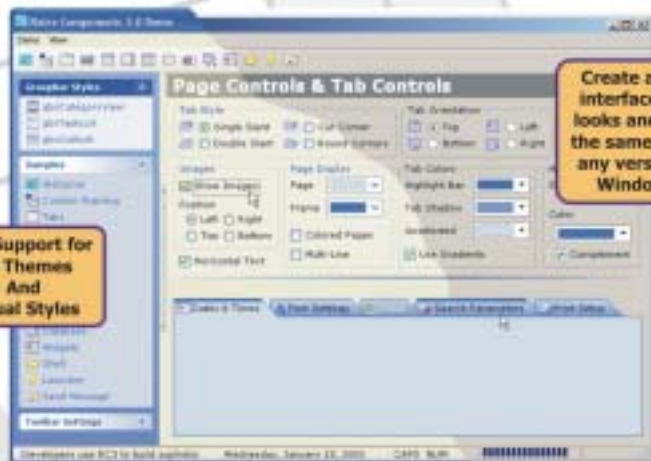
Build sophisticated user interfaces in less time with less effort

**User Interface Design System**  
Raize Components includes **more than 120** general-purpose native VCL controls for use in Delphi and C++Builder.

**Powerful and Easy to Use**  
RC3 also includes **more than 100** component designers focused on simplifying user interface development.

**Component Innovations**  
Raize Components also features one-step installation, automatic help integration, and dynamic component registration.

**Source Code Included**  
Complete source code for all components, packages, and design editors is provided at no additional charge.



Full Support for XP Themes And Visual Styles

Create a user interface that looks and feels the same under any version of Windows.

[www.raize.com](http://www.raize.com)

```

function __GetNode(nd : XmlNode; xpath : string) : string;
var Node : XmlNode;
begin
  Node := nd.SelectSingleNode(xpath);
  if Node <> nil then
    __GetNode := Node.InnerText
  else
    __GetNode := '';
end;

```

Now that we know how to load and work with an RSS feed, let's set about building a class that will provide us with a simple, but extensible means of working with RSS feeds.

## Programmatic Handling of an RSS Document

I know that we looked at an RSS feed in part 1 of this series of articles however, for the sake of completeness, here's a snippet of the bug.xml:

```

<?xml version="1.0"?>
<rss version="2.0">
<channel>
  <title>CraigMurphy.com - Developers Group Downloads</title>
  <link>http://www.craigmurphy.com/bug</link>
  <description>I write articles and give presentations for the UK Borland User Group. From
    time to time I need to make source code or PowerPoint slides available - this is
    where to find them.</description>
  <language>en-uk</language>
  <copyright>Copyright 2001-2004 Craig Murphy</copyright>
  <managingEditor>craig.bug.rss@craigmurphy.com</managingEditor>
  <webMaster>craig.bug.rss@craigmurphy.com</webMaster>
  <pubDate>Sun, 7 Mar 2004 22:00:00 GMT</pubDate>
  <lastBuildDate>Sun, 7 Mar 2004 22:00:00 GMT</lastBuildDate>
  <generator></generator>
  <docs>http://blogs.law.harvard.edu/tech/rss</docs>
  <image>
    <title>Craig Murphy.com: Developers Group Downloads</title>
    <link>http://www.craigmurphy.com/</link>
    <url>http://www.craigmurphy.com/images/logo.gif</url>
  </image>

  <item>
    <title>Using OpenXML and the XDOM</title>
    <link>http://www.craigmurphy.com/bug</link>
    <description>Each archive contains the original article as a PDF, source code and GIF
      files for each of the figures &lt;br /&gt;
      &lt;a href="http://www.craigmurphy.com/bug/xdom/OpenXML1.zip"&gt;Download&lt;/a&gt; the Part
      1 (136K): XDOM basics, creating, loading, saving&lt;br /&gt;
      &lt;a href="http://www.craigmurphy.com/bug/xdom/OpenXML2.zip"&gt;Download&lt;/a&gt; the Part
      2 (115K): Traversing XML using a TreeWalker and an Iterator&lt;br /&gt;
      &lt;br /&gt;Thanks are due to the &lt;a href="http://www.richplum.co.uk/"&gt;Developers
      Group&lt;/a&gt; for their kind permission to reproduce these here.
    </description>
    <author>craig.bug.rss@craigmurphy.com</author>
    <comments>http://www.craigmurphy.com/bug</comments>
    <pubDate>Sun, 7 Mar 2004 22:00:00 GMT</pubDate>
  </item>
  ...
</channel>

```

An RSS feed has a root element <channel> that contains a number of elements describing the RSS feed followed by zero or more <item> elements.

We are going to need a means of representing each of the <item> elements. Keeping things simple, TRSSItem goes some way to allowing us to represent an <item> element:

```

TRSSItem = class
  public
    Title,
    Link,
    Description,
    Author,
    Comments,
    PubDate : string;
end;

```

TRSS provides us with a wrapper for the entire RSS feed. It deals with the RSS-specific elements and the variable number of <item> elements. Using the System.Collection namespace, the ArrayList object provides us with a means of working with a collection of objects of any type (although we will stick to items of type TRSSItem).

```
TRSS = class
private
  rssItems : ArrayList;
public
  Title,
  Link,
  Description,
  Language,
  Copyright,
  ManagingEditor,
  WebMaster,
  PubDate,
  LastBuildDate,
  Generator,
  Docs,
  Img,
  ImgTitle,
  ImgLink,
  ImgURL : string;

  constructor Create(FileName : string); overload;
  function SaveFeed(FileName : string) : Boolean;

  function GetRSSItem(itemNo : integer) : TRSSItem;
  function GetRSSItemCount : integer;
  procedure DeleteRSSItem(itemNo : integer);
  procedure AddRSSItem(RSSItem : TRSSItem);
end;
```

TRSS also provides us with a handful of management methods:

Create takes a filename representing an existing RSS feed – it loads the XML document populating an ArrayList containing TRSSItem instances representing the <item> elements.

SaveFeed simply takes the rssItems collection and saves them as an RSS 2.0 feed format, which is nothing more than vanilla XML.

GetRSSItemCount returns the number of <item> elements that were found in the RSS feed.

GetRSSItem takes an integer as a parameter and returns the TRSSItem representing the said <item>.

DeleteRSSItem also takes an integer as a parameter; it deletes the said RSS <item>.

AddRSSItem takes an instance of TRSSItem and adds it to the collection rssItems.

## Using TRSS

Assuming that an RSS feed already exists, TRSS can load it using the following code:

```
Var rss : TRSS;
...
rss := TRSS.Create('bug.xml');
```

There is a lot to the Create method. It has to understand how to work with XML and has to honour the RSS XML format. Luckily, using Delphi 8 and .NET, this is not such a burden. Indeed, .NET's XmlDocument class provides us with access to an XML document's elements and attributes using XPath expressions – this means we can write some rather concise code that accesses the specific RSS elements. We touched on this earlier in this article when we looked at the \_\_GetNode method – more on this later.

Here is the code for the Create method:

```
constructor TRSS.Create(FileName : string);
var xmlDoc : XmlDocument;
  xmlNodes : XmlNodeList;
  Node : XmlNode;
  rssItem : TRSSItem;
  str : string;
  NodeEnum : IEnumerator;
```

```

function __GetNode(nd : XmlNode; xpath : string) : string;
var Node : XmlNode;
begin
  Node := nd.SelectSingleNode(xpath);
  if Node <> nil then __GetNode := Node.InnerText
  else __GetNode := '';
end;

begin
  inherited Create;

  xmlDoc := XmlDocument.Create;
  xmlDoc.Load(FileName);

  Title      := __GetNode(xmlDoc, 'rss/channel/title');
  Link       := __GetNode(xmlDoc, 'rss/channel/link');
  Description := __GetNode(xmlDoc, 'rss/channel/description');
  Language   := __GetNode(xmlDoc, 'rss/channel/language');
  Copyright  := __GetNode(xmlDoc, 'rss/channel/copyright');
  ManagingEditor := __GetNode(xmlDoc, 'rss/channel/managingEditor');
  Language   := __GetNode(xmlDoc, 'rss/channel/language');
  WebMaster  := __GetNode(xmlDoc, 'rss/channel/webMaster');
  PubDate    := __GetNode(xmlDoc, 'rss/channel/pubDate');
  LastBuildDate := __GetNode(xmlDoc, 'rss/channel/lastBuildDate');
  Generator  := __GetNode(xmlDoc, 'rss/channel/generator');
  Docs       := __GetNode(xmlDoc, 'rss/channel/docs');
  ImgTitle   := __GetNode(xmlDoc, 'rss/channel/image/title');
  ImgLink    := __GetNode(xmlDoc, 'rss/channel/image/link');
  ImgURL     := __GetNode(xmlDoc, 'rss/channel/image/url');

  rssItems := ArrayList.Create;

  xmlNodes := xmlDoc.SelectNodes('rss/channel/item');

  NodeEnum := xmlNodes.GetEnumerator;
  while NodeEnum.MoveNext() do begin
    rssItem := TRSSItem.Create;
    Node := NodeEnum.Current As XmlNode;

    str:=Node.OuterXml;

    rssItem.Title      := __GetNode(Node, 'title');
    rssItem.Link       := __GetNode(Node, 'link');
    rssItem.Description := __GetNode(Node, 'description');
    rssItem.Author     := __GetNode(Node, 'author');
    rssItem.Comments   := __GetNode(Node, 'comments');
    rssItem.PubDate    := __GetNode(Node, 'pubDate');

    rssItems.Add(rssItem);
  end;
end;

```

SelectSingleNode is a useful function (specific to MSXML and the .NET framework) that takes XPath expressions to allow us essentially to get the information held in specific nodes. You can see that I have wrapped its functionality up in the \_\_GetNode method (whenever I write a method that is local to another method I tend to prefix the method with a double underscore).

Create also uses an enumerator to iterate over the <item> nodes found in the RSS feed. Whilst we had access to enumerators in Win32 (pre-Delphi 8) versions of Delphi, their use was not as commonplace as it is/will be in Delphi 8, largely because of their extensive usage in the .NET framework. It is likely that heavy users of COM in a Win32 environment will be familiar with the use of enumerators. As you can see from the while loop in the Create method, their .NET syntax is pleasant and not as littered as their Win32 counterpart.

## Iterating Over RSSItems

The overloaded TRSS Create method builds an ArrayList of TRSSItems representing the <item> elements found in the RSS feed. Typically we will want to iterate over that collection, possibly adding them to a ListView for example. The following code snippet demonstrates how we might achieve this:

```

var colNo, itemNo : integer;
    rssItem : TRSSItem;
    lvItem : ListViewItem;
...

```

```

lvRSS.Items.Clear;
for itemNo := 0 to rss.GetRSSItemCount - 1 do begin
  rssItem := rss.GetRSSItem(itemNo);

  lvItem := lvRSS.Items.Add(rssItem.Title);
  lvItem.SubItems.Add(rssItem.PubDate);
end;

```

## Adding New <items>

```

procedure TRSS.AddRSSItem(rssItem: TRSSItem);
var newrssItem : TRSSItem;
begin
  newrssItem := TRSSItem.Create;

  newrssItem.Title      := rssItem.Title;
  newrssItem.Link       := rssItem.Link;
  newrssItem.Description := rssItem.Description;
  newrssItem.Author     := rssItem.Author;
  newrssItem.Comments   := rssItem.Comments;
  newrssItem.PubDate    := rssItem.PubDate;

  rssItems.Insert(0, newrssItem);
end;

```

## Saving an RSS Feed

Now that we have seen how to add new RSS items, we need to be able to save the RSS items for subsequent upload to a website. Like the load (Create) method, the SaveFeed method has to know how to work with .NET's XmlDocument class and know about the RSS 2.0 XML format.

I have kept the implementation of the SaveFeed method fairly straightforward, preferring to get it working before I strengthen it and refactor it into something a little more elegant. Its operation is perhaps obvious: create an XML document, create the RSS elements <rss>, <channel> and an <item> for each item in the rssItems collection. For good measure, this method relies on its own internal properties and methods, notably, GetRSSItem and GetRSSItemCount.

```

function TRSS.SaveFeed(FileName: string): Boolean;
var xmlDoc : XmlDocument;
  xmlDec : XmlDeclaration;
  xmlRoot, newNode, channelNode, imageNode : XmlNode;
  xmlAtt : XmlAttribute;
  itemNo : integer;

  procedure __AddElement(destNode : XmlNode; NodeName, NodeContent : string);
  begin
    newNode := xmlDoc.CreateNode(XmlNodeType.Element, NodeName, '');
    newNode.InnerText := NodeContent;
    destNode.AppendChild(newNode);
  end;

  procedure __AddRSSItem(rssItem : TRSSItem);
  var itemNode : XmlNode;
  begin
    itemNode := xmlDoc.CreateNode(XmlNodeType.Element, 'item', '');
    __AddElement(itemNode, 'title',      rssItem.Title);
    __AddElement(itemNode, 'link',       rssItem.Link);
    __AddElement(itemNode, 'description', rssItem.Description);
    __AddElement(itemNode, 'author',     rssItem.Author);
    __AddElement(itemNode, 'pubDate',    rssItem.PubDate);
    channelNode.AppendChild(itemNode);
  end;

begin
  xmlDoc := XmlDocument.Create;

  xmlDec := xmlDoc.CreateXmlDeclaration('1.0', '', '');
  xmlDoc.AppendChild(xmlDec);

  xmlRoot := xmlDoc.CreateElement('rss');
  xmlAtt := xmlDoc.CreateAttribute('version');
  xmlAtt.InnerText := '2.0';
  xmlRoot.Attributes.Append(xmlAtt);

```

```

xmlDoc.AppendChild(xmlRoot);

channelNode := xmlDoc.CreateNode(XmlNodeType.Element, 'channel', '');

__AddElement(channelNode, 'title', Title);
__AddElement(channelNode, 'link', Link);
__AddElement(channelNode, 'description', Description);
__AddElement(channelNode, 'language', Language);
__AddElement(channelNode, 'copyright', Copyright);
__AddElement(channelNode, 'managingEditor', ManagingEditor);
__AddElement(channelNode, 'webMaster', WebMaster);
__AddElement(channelNode, 'pubDate', PubDate);
__AddElement(channelNode, 'generator', Generator);
__AddElement(channelNode, 'docs', Docs);

imageNode := xmlDoc.CreateNode(XmlNodeType.Element, 'image', '');
__AddElement(imageNode, 'title', ImgTitle);
__AddElement(imageNode, 'link', ImgLink);
__AddElement(imageNode, 'url', ImgURL);
channelNode.AppendChild(imageNode);

for itemNo := 0 to GetRSSItemCount - 1 do
  __AddRSSItem( GetRSSItem(itemNo) );

xmlRoot.AppendChild(channelNode);

xmlDoc.Save(FileName);

SaveFeed := True;
end;

```

Listing 1 presents the TRSS class that has been discussed here. If you visit <http://www.craigmurphy.com/bug> you will be able to download the full source code for a simple RSS management application. It is this application that I will be covering in forthcoming articles, so expect the application to grow arms and legs over the coming months.

## Summary

Over the course of this article I have demonstrated how we can use Delphi 8 to work with an RSS formatted XML document. Whilst we didn't cover very much ground, we did see how to load and save XML documents using Delphi 8. We also have the bare bones of a reusable RSS class: we can create an RSS feed/file, add items, delete items and save the feed as an XML file.

In my next article I will add some gloss to this application – after all, the code presented here is far from perfect! We'll see how to generate RSS feeds from scratch and we'll take a look at how we can augment this application to handle more than one RSS feed. I deliberately glossed over some important aspects of RSS, such as the RFC822 date format. All RSS dates must adhere to this standard and whilst the code in this article matches that standard, it's not very flexible.

Also in the next article will be more coverage about how to include HTML tags in an RSS feed without the need to “escape” them using “&lt;” and “&gt;”.

For those of you who are already blogging (writing a web log/diary), I am sure it has not escaped your attention that many blogs are served up via an RSS feed. To that end, you will find a few notable blogs listed at the end of this article.

I know that I am repeating myself, but I am finding that RSS is an excellent way of staying on top of things: news comes to me rather than me having to seek out the news. Elsewhere in this publication you will find me rambling on about my e-mail/link (favourites/bookmarks) manager/newsgroups applications that I use. I discussed a product called LinkMan – it has a “Daily Links” feature which provides a dumping ground for URLs that I would like to visit daily (or more realistically, frequently). It is still a chore visiting a handful of sites each day to see if there is new content. If there is an RSS feed, let the new content come to you!

## Lastly...

As I write this I learn of the untimely death of friend and colleague Jon Jenkinson. I had the pleasure and privilege of essentially being mentored by Jon during my early presentations at BUG meetings. My first ever article for the BUG magazine was reviewed by Jon – his positive (and negative!) feedback has shaped and influenced my writing style – permanently.

Jon was also a catalyst in my presence at DCon 2001 as a “yellow shirt”. For some reason this is my most fond memory of Jon: a yellow shirt, black trousers, bouncing from room to room, sneaking the occasional fag in between times!

Jon was one of the nicest blokes you could ever expect to meet; he would go out of his way to help and was always there to give a very frank and honest answer to any question posed. It upsets me that I didn't manage to visit Jon and Debra more often – JJ, you are greatly missed.



## Resources

- [1] Implementing RSS using PHP: <http://www.richplum.co.uk/magazine/bug200401/eJanFeb2004.pdf>
  - [2] RSS Aggregators: <http://www.hebig.org/blogs/archives/main/000877.php>
  - [3] A readable version of the RSS 2.0 specification: <http://feedvalidator.org/docs/rss2.html>
  - [4] RSS (and Atom 0.3) feed validation: <http://feedvalidator.org>
- RSSReader (requires .NET): <http://www.rssreader.org>  
FeedReader: <http://www.feedReader.com/>

## RSS News Feeds

- <http://www.richplum.co.uk/DG.xml>
- <http://www.craigmurphy.com/bug/bug.xml>
- <http://community.borland.com/article/0,1410,32010,00.html>
- <http://news.borland.com/>

## Notable Blogs

- Anders Ohlsson: [http://homepages.borland.com/aohlsson/blog\\_beta/index.html](http://homepages.borland.com/aohlsson/blog_beta/index.html)
- Allen Bauer: <http://homepages.borland.com/abauer/>

---

*Craig is an author, developer, speaker, project manager and is a Certified ScrumMaster. He specialises in all things XML, particularly SOAP and XSLT. Craig is evangelical about .NET, C#, Test-Driven Development, Extreme Programming, agile methods and Scrum. He can be reached via e-mail at: [bug@craigmurphy.com](mailto:bug@craigmurphy.com), or via his web site: <http://www.craigmurphy.com> (where you can also find the source code and PowerPoint files for all of Craig's articles, reviews and presentations).*

### Listing 1 – A simple class that manages an RSS feed

```
unit ClsRSS;

interface

uses System.Collections;

type
  TRSSItem = class
  public
    Title,
    Link,
    Description,
    Author,
    Comments,
    PubDate : string;
  end;

  TRSS = class
  private
    rssItems : ArrayList;
  public
    Title,
    Link,
    Description,
    Language,
    Copyright,
```

```

    ManagingEditor,
    WebMaster,
    PubDate,
    LastBuildDate,
    Generator,
    Docs,
    Img,
    ImgTitle,
    ImgLink,
    ImgURL : string;

    constructor Create(FileName : string); overload;
    function SaveFeed(FileName : string) : Boolean;

    function GetRSSItem(itemNo : integer) : TRSSItem;
    function GetRSSItemCount : integer;
    procedure DeleteRSSItem(itemNo : integer);
    procedure AddRSSItem(RSSItem : TRSSItem);
    procedure UpdateRSSItem(itemNo : integer; RSSItem : TRSSItem);
end;

implementation

uses System.Xml;

(* _____ *)

procedure TRSS.AddRSSItem(rssItem: TRSSItem);
var newrssItem : TRSSItem;
begin
    newrssItem := TRSSItem.Create;

    newrssItem.Title := rssItem.Title;
    newrssItem.Link := rssItem.Link;
    newrssItem.Description := rssItem.Description;
    newrssItem.Author := rssItem.Author;
    newrssItem.Comments := rssItem.Comments;
    newrssItem.PubDate := rssItem.PubDate;

    rssItems.Insert(0, newrssItem);
end;

(* _____ *)

constructor TRSS.Create(FileName : string);
var xmlDoc : XmlDocument;
xmlNodes : XmlNodeList;
Node : XmlNode;
rssItem : TRSSItem;
str : string;
NodeEnum : IEnumerator;

function __GetNode(nd : XmlNode; xpath : string) : string;
var Node : XmlNode;
begin
    Node := nd.SelectSingleNode(xpath);
    if Node <> nil then __GetNode := Node.InnerText
    else __GetNode := '';
end;

begin
    inherited Create;

    xmlDoc := XmlDocument.Create;
    xmlDoc.Load(FileName);

    Title := __GetNode(xmlDoc, 'rss/channel/title');
    Link := __GetNode(xmlDoc, 'rss/channel/link');
    Description := __GetNode(xmlDoc, 'rss/channel/description');
    Language := __GetNode(xmlDoc, 'rss/channel/language');
    Copyright := __GetNode(xmlDoc, 'rss/channel/copyright');
    ManagingEditor := __GetNode(xmlDoc, 'rss/channel/managingEditor');
    Language := __GetNode(xmlDoc, 'rss/channel/language');
    WebMaster := __GetNode(xmlDoc, 'rss/channel/webMaster');
    PubDate := __GetNode(xmlDoc, 'rss/channel/pubDate');
    LastBuildDate := __GetNode(xmlDoc, 'rss/channel/lastBuildDate');

```

```

Generator      := __GetNode(xmlDoc, 'rss/channel/generator');
Docs           := __GetNode(xmlDoc, 'rss/channel/docs');
ImgTitle       := __GetNode(xmlDoc, 'rss/channel/image/title');
ImgLink        := __GetNode(xmlDoc, 'rss/channel/image/link');
ImgURL         := __GetNode(xmlDoc, 'rss/channel/image/url');

rssItems := ArrayList.Create;

xmlNodes := xmlDoc.SelectNodes('rss/channel/item');

NodeEnum := xmlNodes.GetEnumerator;
while NodeEnum.MoveNext() do begin
    rssItem := TRSSItem.Create;
    Node := NodeEnum.Current As XmlNode;

    str:=Node.OuterXml;

    rssItem.Title      := __GetNode(Node, 'title');
    rssItem.Link       := __GetNode(Node, 'link');
    rssItem.Description := __GetNode(Node, 'description');
    rssItem.Author     := __GetNode(Node, 'author');
    rssItem.Comments   := __GetNode(Node, 'comments');
    rssItem.PubDate    := __GetNode(Node, 'pubDate');

    rssItems.Add(rssItem);
end;
end;

(* _____ *)

procedure TRSS.DeleteRSSItem(itemNo: integer);
begin
    rssItems.RemoveAt(itemNo);
end;

(* _____ *)

function TRSS.GetRSSItem(itemNo: integer): TRSSItem;
begin
    GetRSSItem := rssItems.Item[itemNo] As TRSSItem;
end;

(* _____ *)

function TRSS.GetRSSItemCount: integer;
begin
    GetRSSItemCount := rssItems.Count;
end;

(* _____ *)

function TRSS.SaveFeed(FileName: string): Boolean;
var xmlDoc : XmlDocument;
    xmlDec : XmlDeclaration;
    xmlRoot, newNode, channelNode, imageNode : XmlNode;
    xmlAtt : XmlAttribute;
    itemNo : integer;

procedure __AddElement(destNode : XmlNode; NodeName, NodeContent : string);
begin
    newNode          := xmlDoc.CreateNode(XmlNodeType.Element, NodeName, '');
    newNode.InnerText := NodeContent;
    destNode.AppendChild(newNode);
end;

procedure __AddRSSItem(rssItem : TRSSItem);
var itemNode : XmlNode;
begin
    itemNode := xmlDoc.CreateNode(XmlNodeType.Element, 'item', '');
    __AddElement(itemNode, 'title',      rssItem.Title);
    __AddElement(itemNode, 'link',       rssItem.Link);
    __AddElement(itemNode, 'description', rssItem.Description);
    __AddElement(itemNode, 'author',     rssItem.Author);
    __AddElement(itemNode, 'pubDate',    rssItem.PubDate);
    channelNode.AppendChild(itemNode)
end;
end;

```

```

begin
  xmlDoc := XmlDocument.Create;

  xmlDec := xmlDoc.CreateXmlDeclaration('1.0', '', '');
  xmlDoc.AppendChild(xmlDec);

  xmlRoot := xmlDoc.CreateElement('rss');
  xmlAtt := xmlDoc.CreateAttribute('version');
  xmlAtt.InnerText := '2.0';
  xmlRoot.Attributes.Append(xmlAtt);
  xmlDoc.AppendChild(xmlRoot);

  channelNode := xmlDoc.CreateNode(XmlNodeType.Element, 'channel', '');

  __AddElement(channelNode, 'title', Title);
  __AddElement(channelNode, 'link', Link);
  __AddElement(channelNode, 'description', Description);
  __AddElement(channelNode, 'language', Language);
  __AddElement(channelNode, 'copyright', Copyright);
  __AddElement(channelNode, 'managingEditor', ManagingEditor);
  __AddElement(channelNode, 'webMaster', WebMaster);
  __AddElement(channelNode, 'pubDate', PubDate);
  __AddElement(channelNode, 'generator', Generator);
  __AddElement(channelNode, 'docs', Docs);

  imageNode := xmlDoc.CreateNode(XmlNodeType.Element, 'image', '');
  __AddElement(imageNode, 'title', ImgTitle);
  __AddElement(imageNode, 'link', ImgLink);
  __AddElement(imageNode, 'url', ImgURL);
  channelNode.AppendChild(imageNode);

  for itemNo := 0 to GetRSSItemCount - 1 do
    __AddRSSItem( GetRSSItem(itemNo) );

  xmlRoot.AppendChild(channelNode);

  xmlDoc.Save(FileName);

  SaveFeed := True;
end;

(* _____ *)

procedure TRSS.UpdateRSSItem(itemNo: integer; RSSItem: TRSSItem);
begin
  with TRSSItem(rssItems[itemNo]) do begin
    Title := RSSItem.Title;
    Link := RSSItem.Link;
    Description := RSSItem.Description;
    Author := RSSItem.Author;
    Comments := RSSItem.Comments;
    PubDate := RSSItem.PubDate;
  end;
end;

end.

```

**Listing 1 – A simple class that manages an RSS feed**

