Integrating RSS and Visual FoxPro

Doug Hennig
Stonefield Software Inc.
Email: dhennig@stonefield.com
Web site: http://www.stonefield.com
Web site: http://www.stonefieldquery.com
Blog: http://doughennig.blogspot.com

Overview

RSS (Really Simple Syndication) is becoming a popular way to publish content on Web sites and blogs. However, it can be used for a lot more than that. This document provides an introduction to RSS and explores how VFP can both generate and consume RSS.

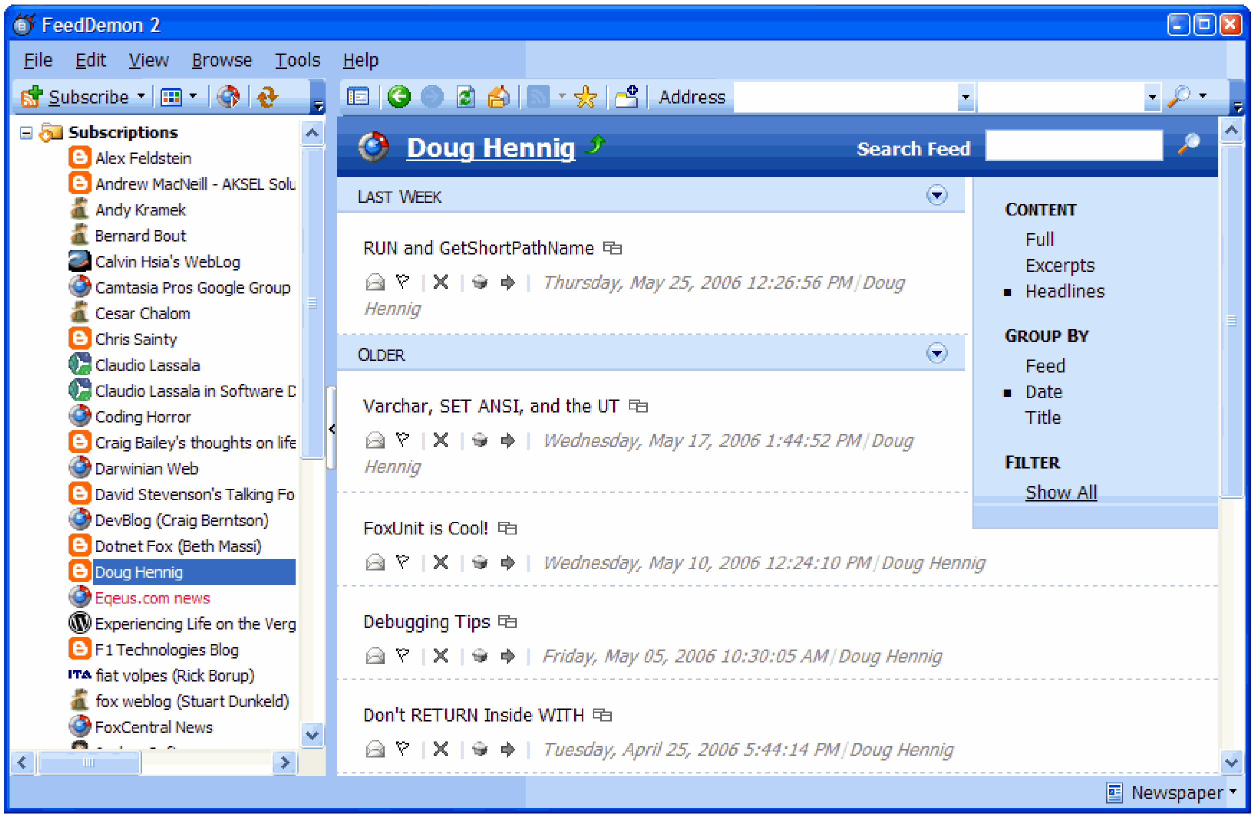
Topics I'll discuss include:

- What RSS is.
- The structure of RSS feeds.
- How to consume RSS in VFP.
- How to generate RSS from VFP.
- Uses of RSS in VFP applications.

What is RSS?

The term “RSS” has had several meanings over the years, such as “Rich Site Summary,” but is now generally understood to stand for “Really Simple Syndication.” RSS is an XML-based mechanism for syndicating and aggregating Web site content. Information providers use RSS to create a data “feed” (an XML file available from a Web site), providing headlines, links, and summarized content. Information consumers use news aggregator software, also known as “feed readers,” to retrieve updated content from these providers.

There are many uses for RSS, the most popular being Web logs (more commonly known as “blogs”) and news syndication. For example, my blog is at http://doughennig.blogspot.com. While you can read it using a Web browser, you can also “subscribe” to its content by telling a feed reader to regularly check the RSS version of my blog (http://doughennig.blogspot.com/atom.xml). You can subscribe to dozens or even thousands of feeds; the feed reader does the work of downloading the XML files from each of the sites you’ve subscribed to and highlighting those that have new or changed items, usually by comparing to a local database of previously read items. **Figure 1** shows a screen shot of a popular feed reader, FeedDemon (http://www.feeddemon.com).



RUN and GetShortPathName
Thursday, May 25, 2006 12:26:56 PM | Doug Hennig

Varchar, SET ANSI, and the UT
Wednesday, May 17, 2006 1:44:52 PM | Doug Hennig

FoxUnit is Cool!
Wednesday, May 10, 2006 12:24:10 PM | Doug Hennig

Debugging Tips
Friday, May 05, 2006 10:30:05 AM | Doug Hennig

Don't RETURN Inside WITH
Tuesday, April 25, 2006 5:44:14 PM | Doug Hennig

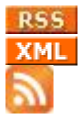
***Figure 1**. Like other feed readers, FeedDemon does all the work of retrieving the latest content from your subscribed feeds.*

There are many benefits to RSS:

- It’s automatically an opt-in mechanism: you only subscribe to feeds containing information you’re interested in.
- RSS is vastly superior to other mechanisms for publishing content. Email blasts, for example, consume resources on a server, require a mailing list that must be managed, may have legal issues, and can make you look like a spammer to your ISP. Also, an email may not be read by the intended party due to anti-spam blocking or simple “inbox fatigue.”
- RSS is a pull rather than push technology. Providers publish an XML file on their Web site and interested subscribers retrieve a copy of that file at their convenience.

- It allows subscribers to view some Web site content from publishers without actually visiting their Web sites. For example, if you want to see what's new at 50 different companies you do business with, you could manually navigate to 50 different Web sites and try to hunt down the information, or you could subscribe to their feeds and let a feed reader do all the work.
- RSS feeds can be used for more than simply viewing content on a Web site. For example, many feed readers allow you to cross-search multiple feeds at the same time.

Sites that feature RSS feeds often provide an image you can click to retrieve the feed. This image is usually one of the following:



The third image is being promoted as the new standard and many variations of it are available at http://www.feedicons.com.

There are a lot of resources available if you wish to learn more about the basics of RSS. A simple one in layman's terms is available at http://www.nytimes.com/ref/technology/circuits/03basi.html.

The Structure of RSS Feeds

An RSS feed is simply an XML file with a particular schema available for download from a Web site. However, there are several competing specifications for the structure of RSS files. The two most common ones are RSS 2.0 and Atom 1.0. These specifications differ significantly, but most feed readers can understand multiple formats, so it isn't generally a problem for consumers. In this document, I'll focus on RSS 2.0.

An RSS file contains two types of information:

- Information about the feed itself, such as the title, description, date it was last updated, and a link to a Web site. RSS 2.0 refers to this as a "channel," while Atom 1.0 calls it a "feed."
- Information about the articles or items in the feed, such as the title, description, date it was published, and a link to a Web site. The description may be the full content of the item or just a simple summary of the content, in which case the link for the item references the full content. RSS 2.0 calls these components "items;" Atom 1.0 calls them "entries."

An RSS 2.0 file looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0">
  <channel>
    channel elements
    <item>
      item elements
    </item>
  </channel>
</rss>
```

Although there's a single channel element, there can be multiple item elements. Here's an example:

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0">
<channel>
  <title>White Papers by Doug Hennig</title>
  <link>http://www.stonefield.com/techpap.html</link>
  <description>Doug Hennig has written a number of white papers, available from the Technical
Papers page of www.stonefield.com.</description>
  <lastBuildDate>Sun, 28 May 2006 20:48:40 GMT</lastBuildDate>
  <item>
    <title>Object-Oriented Menus</title>
    <description>This article, originally published in the August 2001 issue of FoxTalk,
describes how to create object-oriented menus in VFP.</description>
    <link>http://www.stonefield.com/articles/ft/200108dhen.html</link>
    <author>dhennig@stonefield.com (Doug Hennig)</author>
    <guid isPermaLink="false">_1TA0UHHOB</guid>
    <pubDate>Sat, 13 Apr 2002 06:00:00 GMT</pubDate>
  </item>
  <item>
    <title>Report Objects</title>
    <description>This article describes how a set of classes can expose the VFP report (FRX) file
as objects. This allows you to manipulate reports programmatically simply by setting properties
and calling methods.</description>
    <link>http://downloads.stonefield.com/pub/repobj.html</link>
    <author>dhennig@stonefield.com (Doug Hennig)</author>
    <guid isPermaLink="false">_1TA0UHQF7</guid>
    <pubDate>Wed, 12 Apr 2000 06:00:00 GMT</pubDate>
  </item>
</channel>
</rss>
```

A complete specification for RSS 2.0 files is available at http://www.rssboard.org/rss-specification; for Atom, see http://www.atomenabled.org/developers/syndication/atom-format-spec.php. **Table 1** shows the most commonly used channel elements and **Table 2** shows those for items.

***Table 1**. Most commonly used channel elements.*

Element	Purpose
title	The name of the channel. Required.
description	The description of the channel. Required.
link	The URL for the Web site related to the channel. Required.
lastBuildDate	The last time the content of the channel was changed. Optional.

***Table 2**. Most commonly used item elements.*

Element	Purpose
title	The title of the item. Optional, but either title or description must be specified.
description	The item summary or the full content.
link	The URL for the full content (may be omitted or blank if description contains the complete content).
author	The email address of the author.
guid	A string that uniquely identifies the item. Aggregators use this to determine if the item is new. The isPermaLink attribute, which is optional but defaults to “true” if omitted, indicates whether the guid is a real URL.
pubDate	The date and time the item was published.

Note that dates are expressed in RFC 822 format, such as Sun, 28 May 2006 20:48:40 GMT. Also, additional elements can be added, such as those in the “dc,” “slash,” and “wfw” namespaces in this sample taken from Andy Kramek’s blog (http://weblogs.foxite.com/andykramek/rss.aspx). These elements may or may not be understood by a particular feed reader.

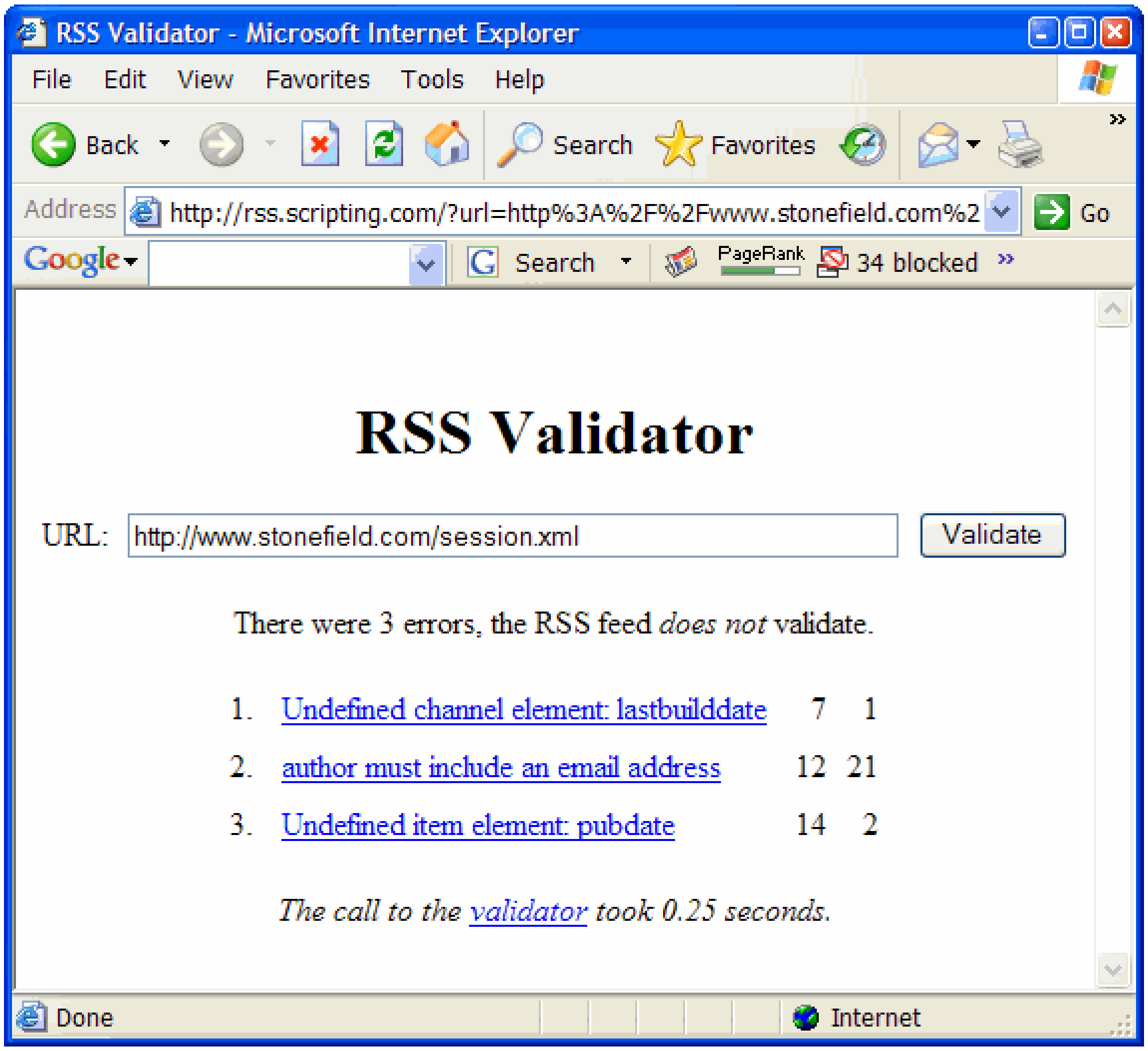
```
<rss version="2.0"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:slash="http://purl.org/rss/1.0/modules/slash/"
xmlns:wfw="http://wellformedweb.org/CommentAPI/">
  <channel>
    ...other elements...
    <dc:language>en-US</dc:language>
    <item>
      ...other elements...
      <dc:creator>andykr</dc:creator>
      <slash:comments>11</slash:comments>
      <wfw:commentRss>http://weblogs.foxite.com/andykramek/commentrss.aspx</wfw:commentRss>
    </item>
  </channel>
</rss>
```

Validating RSS

If you've ever tried to subscribe to an RSS feed using a feed reader and gotten an error, it's likely because the RSS file wasn't valid. First, the feed must conform to the usual rules that apply to XML: there must be a single root element, elements must be properly nested and have closing tags, attributes must be enclosed in quotes, and so forth. Also, remember that XML is case-sensitive, so you must specify elements and attributes in the correct case, and that certain characters must be encoded, or translated to their XML equivalent; for example, "&" must be changed to "&amp;".

Second, the feed must conform to the RSS specification you're using. For example, for RSS 2.0 files, there must be a single channel element, it must have title, description, and link elements, item elements must be nested under channel elements, and so forth.

There are several Web sites that validate RSS, including http://rss.scripting.com and http://www.feedvalidator.org. Both of these require that you provide a URL to a publicly accessible RSS file. **Figure 2** shows the result of validating an RSS file which didn't respect the case of the lastBuildDate and pubDate elements and specified the author's name as a name rather than an email address.


RSS Validator - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://rss.scripting.com/?url=http%3A%2F%2Fwww.stonefield.com%2 Go
Google Search PageRank 34 blocked
Done Internet

RSS Validator

URL: http://www.stonefield.com/session.xml Validate

There were 3 errors, the RSS feed *does not* validate.

1. Undefined channel element: lastbuilddate 7 1
2. author must include an email address 12 21
3. Undefined item element: pubdate 14 2

The call to the validator took 0.25 seconds.

Figure 2*. The result of validating an invalid RSS file at rss.scripting.com.*

Updating RSS

To publish new items to a feed, simply post an updated RSS file. You have a couple of choices to do this: regenerating the entire RSS file or updating it.

If you regenerate the file, you have to ensure the pubDate and guid elements of existing items are preserved or else they'll appear as new items to a feed reader. This isn't a problem if these values are stored with the information for the item but can be if they're generated on the fly by the generator software. One of the benefits of regenerating the file, though, is the possibility of removing outdated items at the same time.

To update the file, insert new items into the existing XML. This means parsing the XML to find the appropriate place to put it, but that isn't difficult to do.

RSS Best Practices

Here are some best practices to consider when generating your own RSS:

- Don't use HTML in the RSS (it's fine in content linked to by the RSS) because it requires the client reader to parse it and makes it harder to use for indexable meta data.
- Use unique values in the guid attribute to prevent duplicates and ensure feed readers can identify new items.
- Use good titles so consumers can surf for items of interest.
- Put this in the header of a Web page that links to an RSS feed so the RSS icon in the browser toolbar will be enabled:

```
<link rel="alternate" type="application/rss+xml" title="Name of feed" href="NameOfFile.XML">
```

- Validate the RSS to ensure it can be consumed correctly.

Generating RSS with VFP

Because RSS files contain XML, and XML is really just formatted text, and VFP excels at manipulating text, generating RSS from VFP is pretty easy. To make it even easier, we'll use a set of wrapper classes to do the hard work.

SFRSSGenerator

SFRSSGenerator, in SFRSS.VCX, is a subclass of Custom. Because there are several specifications for RSS, this class is abstract; it provides an interface for generating RSS, but the actual implementation is left for specification-specific subclasses. The properties of SFRSSGenerator, most of which map to channel elements, are shown in **Table 3**.

***Table 3**. The properties of SFRSSGenerator.*

Element	Purpose
Description	The feed description.
Items	A collection of items for the feed.
LastBuildDate	The last time the content of the feed was changed
Link	The link for the feed.
Title	The title of the feed.

Because a channel can consist of many items, the Init method instantiates a Collection object into the Items property. The AddItem method adds an instance of SFRSSItem, also defined in SFRSS.VCX, to the Items collection; we'll look at SFRSSItem later.

The GetRSS method is the main method; it generates and returns the RSS from the properties of the object and the Items collection. As you can see, this method is pretty simple: it calls the StartFeed method to generate the header and channel XML, calls GetRSSForItem to generate the RSS for each item in the collection, and finally calls EndFeed to finish off the XML. Those methods are all abstract in this class.

```
local lcRSS, ;
  loItem
lcRSS = This.StartFeed()
for each loItem in This.Items
  lcRSS = lcRSS + This.GetRSSForItem(loItem)
next loItem
lcRSS = lcRSS + This.EndFeed()
return lcRSS
```

SFRSSGenerator has another method, GetFormattedDate, which returns a DateTime value in the appropriate RSS format. It too is abstract since different specifications use different formats.

The Load method loads the XML from an existing RSS file, which is useful if you want to add new items to an existing file and then regenerate the RSS feed. It starts by validating the passed XML. It then reads the properties of the feed itself from the XML, obtains a collection of items in the feed, and reads the properties for each one, adding them to the class' collection. All of the methods called by Load are abstract in this class.

```
lparameters tcXML
local llReturn, ;
  loItems, ;
  loItem, ;
  loNewItem

* Ensure valid XML was specified.

if vartype(tcXML) = 'C' and not empty(tcXML)
  with This
    llReturn = .ValidateRSS(tcXML)
    if llReturn

* Get the properties of the feed, then a collection of items. For each item,
* create a new SFRSSItem object in our collection and set its properties.

      .GetFeedProperties(tcXML)
      loItems = .GetItems(tcXML)
      for each loItem in loItems
        loNewItem = .AddItem()
        .GetItemProperties(loItem, loNewItem)
      next loItem
    endif llReturn
  endwith
endif vartype(tcXML) = 'C' ...
return llReturn
```

SFRSSGeneratorRSS2

SFRSSGeneratorRSS2 is a subclass of SFRSSGenerator that generates XML using the RSS 2.0 specification. Although I didn't create one, you could create a subclass of SFRSSGenerator for Atom 1.0 or any other specification you wish to use.

Its Init method determines the offset between the current time zone and GMT (Greenwich Mean Time), because that's needed by GetFormattedDate.

```
local lcTimeZone, ;
  lnID, ;
  lnStandardOffset, ;
  lnDaylightOffset

* Do the usual behavior.

dodefault()

* Declare the time zone information API function and get the time zone
* information.

#define TIME_ZONE_SIZE  172
declare integer GetTimeZoneInformation in kernel32 ;
  string @lpTimeZoneInformation
lcTimeZone = replicate(chr(0), TIME_ZONE_SIZE)
```

```
lnID       = GetTimeZoneInformation(@lcTimeZone)

* Determine the standard and daylight time offset.

lnStandardOffset = ctobin(substr(lcTimeZone,   1, 4), '4RS')
lnDaylightOffset = ctobin(substr(lcTimeZone, 169, 4), '4RS')

* Determine the total offset based on whether the computer is on daylight time
* or not.

if lnID = 2  && daylight time
  This.nTimeZoneOffset = (lnStandardOffset + lnDaylightOffset) * 60
else   && standard time
  This.nTimeZoneOffset = lnStandardOffset * 60
endif lnID = 2
```

StartFeed generates the XML for the header and channel XML.

```
local ldDate, ;
  lcRSS
ldDate = evl(This.LastBuildDate, datetime())
text to lcRSS textmerge noshow
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0">
<channel>
  <title><<This.Title>></title>
  <link><<This.Link>></link>
  <description><<This.Description>></description>
  <lastBuildDate><<This.GetFormattedDate(ldDate)>></lastBuildDate>

endtext
return lcRSS
```

GetRSSForItem generates the XML for the specified SFRSSItem object. This method assumes the isPermaLink attribute of the guid element should be false; feel free to modify this code if you want it to be true or want to specify which value to use. Also, the code uses SYS(2015) if the GUID property of the SFRSSItem object is empty and the current date and time if PublicationDate isn't specified.

```
lparameters toItem as SFRSSItem of SFRSS.VCX
local ldDate, ;
  lcGUID, ;
  lcRSS
ldDate = evl(toItem.PublicationDate, datetime())
lcGUID = evl(toItem.GUID, sys(2015))
text to lcRSS textmerge noshow
  <item>
    <title><<toItem.Title>></title>
    <description><<toItem.Description>></description>
    <link><<toItem.Link>></link>
    <author><<toItem.Author>></author>
    <guid isPermaLink="false"><<lcGUID>></guid>
    <pubDate><<This.GetFormattedDate(ldDate)>></pubDate>
  </item>

endtext
return lcRSS
```

EndFeed finishes off the XML.

```
local lcRSS
text to lcRSS noshow
</channel>
</rss>

endtext
return lcRSS
```

GetFormattedDate returns the specified DateTime value as an RFC 822 string.

```
lparameters ttDate
local ltDate, ;
  lcDateString
ltDate       = ttDate + This.nTimeZoneOffset
lcDateString = left(cdow(ltDate), 3) + ', ' + ;
  padl(day(ltDate), 2, '0') + ' ' + ;
  left(cmonth(ltDate), 3) + ' ' + ;
  transform(year(ltDate)) + ' ' + ;
  padl(hour(ltDate), 2, '0') + ':' + ;
  padl(minute(ltDate), 2, '0') + ':' + ;
  padl(sec(ltDate), 2, '0') + ;
  ' GMT'
return lcDateString
```

ValidateRSS, called from Load to validate the specified XML, instantiates an MS XML DOM object into the custom oXML property and loads the specified XML. Note that it doesn't check to ensure it conforms to the RSS 2.0 specification, although that would be a useful thing to do.

```
lparameters tcXML
local llReturn
with This
  .oXML = createobject('MSXML2.DOMDocument.4.0')
  .oXML.async = .F.
  .oXML.loadXML(tcXML)
  llReturn = .oXML.parseError.errorCode = 0
endwith
return llReturn
```

GetFeedProperties uses the XML DOM object to parse the XML into the appropriate properties, and calls the RFC822ToDateTime method to convert the RFC 822-formatted date into a VFP DateTime value.

```
lparameters tcXML
local loNode, ;
  lcLastBuild
with This
  loNode         = .oXML.selectSingleNode('//channel')
  .Title         = loNode.selectSingleNode('title').text
  .Description   = loNode.selectSingleNode('description').text
  .Link          = loNode.selectSingleNode('link').text
  lcLastBuild    = loNode.selectSingleNode('lastBuildDate').text
  .LastBuildDate = .RFC822ToDateTime(lcLastBuild)
endwith
```

GetItems is simple: it asks the XML DOM object to retrieve the collection of item nodes from the XML.

```
Lparameters tcXML
local loNodes
loNodes = This.oXML.selectNodes('//channel/item')
return loNodes
```

Like GetFeedProperties, GetItemProperties uses the XML DOM object to parse the XML for the specified item node into properties.

```
lparameters toItem, ;
  toNewItem
local lcPubDate
with toNewItem
  .Title           = toItem.selectSingleNode('title').text
  .Description     = toItem.selectSingleNode('description').text
  .Link            = toItem.selectSingleNode('link').text
  lcPubDate        = toItem.selectSingleNode('pubDate').text
  .PublicationDate = This.RFC822ToDateTime(lcPubDate)
  .Author          = toItem.selectSingleNode('author').text
  .GUID            = toItem.selectSingleNode('guid').text
endwith
```

RFC822ToDateTime does the opposite of GetFormattedDate: it converts an RFC 822 date to a VFP DateTime value.

```
lparameters tcRFC822
local lnI, ;
  laMonths[12], ;
  lcDate, ;
  lnDay, ;
  lcMonth, ;
  lnMonth, ;
  lcYear, ;
  lnYear, ;
  lnHours, ;
  lnMinutes, ;
  lnSeconds, ;
  ltDate

* Create an array of months.

for lnI = 1 to 12
  laMonths[lnI] = left(cmonth(date(2006, lnI, 1)), 3)
next lnI

* Parse the RFC 822 date.

lcDate    = substr(tcRFC822, at(', ', tcRFC822) + 2)
lnDay     = val(strextract(lcDate, '', ' '))
lcMonth   = strextract(lcDate, ' ', ' ', 1)
lnMonth   = ascan(laMonths, lcMonth, -1, -1, 1, 15)
lcYear    = strextract(lcDate, ' ', ' ', 2)
lnYear    = val(lcYear)
lnHours   = val(strextract(lcDate, lcYear + ' ', ':'))
lnMinutes = val(strextract(lcDate, ':', ':'))
lnSeconds = val(strextract(lcDate, ':', ' '))
ltDate    = datetime(lnYear, lnMonth, lnDay, lnHours, lnMinutes, lnSeconds) - ;
  This.nTimeZoneOffset
return ltDate
```

SFRSSItem

SFRSSItem is a Custom-based class which simply contains properties for each item in the feed. Its properties, which map to item elements, are shown in **Table 4**.

Table 4. *The properties of SFRSSItem.*

Element	Purpose
Author	The email address of the author of the item.
Description	The item summary or the full content.
GUID	The GUID for the item.
Link	The URL for the full content.
PublicationDate	The date and time the item was published.
Title	The title of the item.

Generating RSS

To use these wrapper classes, instantiate SFRSSGeneratorRSS2 (or another subclass if you wish), set the properties for the feed, add items to the feed, and set the properties of each. Then call GetRSS to generate the XML for the feed and put the results in an XML file. The GenerateRSS.PRG sample program generates the RSS sample shown earlier.

```
local loRSS as SFRSSGeneratorRSS2 of SFRSS.VCX, ;
  loItem as SFRSSItem of SFRSS.VCX
loRSS = newobject('SFRSSGeneratorRSS2', 'SFRSS.VCX')
```

```
loRSS.Title       = 'White Papers by Doug Hennig'
loRSS.Description = 'Doug Hennig has written a number of white papers, ' + ;
  'available from the Technical Papers page of www.stonefield.com.'
loRSS.Link        = 'http://www.stonefield.com/techpap.html'

loItem = loRSS.AddItem()
loItem.Title           = 'Object-Oriented Menus'
loItem.Description     = 'This article, originally published in the August ' + ;
  '2001 issue of FoxTalk, describes how to create object-oriented menus ' + ;
  'in VFP.'
loItem.Author          = 'dhennig@stonefield.com (Doug Hennig)'
loItem.Link            = 'http://www.stonefield.com/articles/ft/200108dhen.html'
loItem.PublicationDate = {^2002-04-13}

loItem = loRSS.AddItem()
loItem.Title           = 'Report Objects'
loItem.Description     = 'This article describes how a set of classes can ' + ;
  'expose the VFP report (FRX) file as objects. This allows you to ' + ;
  'manipulate reports programmatically simply by setting properties and ' + ;
  'calling methods.'
loItem.Author          = 'dhennig@stonefield.com (Doug Hennig)'
loItem.Link            = 'http://downloads.stonefield.com/pub/repobj.html'
loItem.PublicationDate = {^2000-04-12}

strtofile(loRSS.GetRSS(), 'session.xml')
modify file session.xml nowait
```

Of course, this sample is hard-coded. A data-driven generator would be more useful, as it would generate RSS from a table of content instead. DataDrivenRSS.PRG is an example of that. It generates RSS from two tables, appropriately named CHANNEL.DBF and ITEMS.DBF.

```
close databases all
use CHANNEL
local loRSS as SFRSSGeneratorRSS2 of SFRSS.VCX, ;
  loItem as SFRSSItem of SFRSS.VCX
loRSS = newobject('SFRSSGeneratorRSS2', 'SFRSS.VCX')
loRSS.Title       = alltrim(TITLE)
loRSS.Description = DESCRIP
loRSS.Link        = LINK
use

use ITEMS
scan
  loItem = loRSS.AddItem()
  loItem.Title           = alltrim(TITLE)
  loItem.Description     = DESCRIP
  loItem.Author          = AUTHOR
  loItem.Link            = LINK
  loItem.PublicationDate = PUBDATE
  loItem.GUID            = alltrim(GUID)
endscan
use

strtofile(loRSS.GetRSS(), 'session.xml')
modify file session.xml nowait
```

UpdateRSS.PRG shows how to update an existing RSS file. It calls the Load method of SFRSSGeneratorRSS2 to load the existing file, then adds another item and generates the updated XML.

```
local loRSS as SFRSSGeneratorRSS2 of SFRSS.VCX, ;
  loItem as SFRSSItem of SFRSS.VCX
loRSS = newobject('SFRSSGeneratorRSS2', 'SFRSS.VCX')
loRSS.Load(filetostr('session.xml'))

loItem = loRSS.AddItem()
loItem.Title           = 'N-Tier Application Design'
loItem.Description     = "This is the article and source code for Doug " + ;
  "Hennig's article, N-Tier Application Design, that appeared in the " + ;
  "Summer 2000 issue of Code Magazine."
loItem.Author          = 'dhennig@stonefield.com (Doug Hennig)'
loItem.Link            = 'http://downloads.stonefield.com/pub/ntier.zip'
loItem.PublicationDate = {^2002-03-25}

strtofile(loRSS.GetRSS(), 'session.xml')
modify file session.xml nowait
```

Note that SFRSSGeneratorRSS2 uses text merge to generate the XML. Another approach is to use the Microsoft XML DOM object. One of the benefits of this object is that it automatically handles conversion of certain characters to their escaped values, such as replacing "&" with "&amp;". With SFRSSGeneratorRSS2, that process is manual; you have to do the conversion yourself before storing text in a property. However, working with the XML DOM is a little more challenging: you have to learn the object model, it's harder to add white space to the XML, and it's generally slower, especially as the feed grows larger.

Consuming RSS in VFP

If you need to consume RSS in VFP applications, there are several approaches.

One is to manually parse the XML. The STREXTRACT() function can make short work of reading text between element tags in any XML file, as shown by this code:

```
* Read in the file.

lcXML = filetostr('session.xml')

* Display the channel information.

lcChannel     = strextract(lcXML,     '<channel>',       '</channel>')
lcTitle       = strextract(lcChannel, '<title>',         '</title>')
lcDescription = strextract(lcChannel, '<description>',   '</description>')
lcLink        = strextract(lcChannel, '<link>',          '</link>')
lcLastBuild   = strextract(lcChannel, '<lastBuildDate>', '</lastBuildDate>')
messagebox('Description: ' + lcDescription + chr(13) + ;
  'Link: ' + lcLink + chr(13) + ;
  'Last build date: ' + lcLastBuild, ;
  0, lcTitle)

* Display item information.

for lnI = 1 to occurs('<item>', lcXML)
  lcItem        = strextract(lcXML,  '<item>',        '</item>', lnI)
  lcTitle       = strextract(lcItem, '<title>',       '</title>')
  lcDescription = strextract(lcItem, '<description>', '</description>')
  lcLink        = strextract(lcItem, '<link>',        '</link>')
  lcPubDate     = strextract(lcItem, '<pubDate>',     '</pubDate>')
  lcAuthor      = strextract(lcItem, '<author>',      '</author>')
  lcGUID        = strextract(lcItem, '<guid>',        '</guid>')
  if empty(lcGUID)
    lcGUID = strextract(lcItem, '<guid ', '>', 1, 4)
    lcGUID = strextract(lcItem, lcGUID, '</guid>')
  endif empty(lcGUID)
  messagebox('Description: ' + lcDescription + chr(13) + ;
    'Link: ' + lcLink + chr(13) + ;
    'Published: ' + lcPubDate + chr(13) + ;
    'Author: ' + lcAuthor + chr(13) + ;
    'GUID: ' + lcGUID, ;
    0, lcTitle)
next lnI
```

A couple of downsides to this approach are that it's more work to handle optional attributes, as shown by the way the guid element is treated in the preceding code, and that you have to decode encoded characters manually (for example, using STRTRAN(lcXML, '&amp;', '&') to convert encoded ampersands).

You can also use the MS XML DOM object to handle the work. As we saw earlier, the Load method of SFRSSGeneratorRSS2 already does that, so it's a simple matter to instantiate an instance of it and have it do all the work, as XMLDOMParse.PRG shows:

```
* Instantiate an SFRSSGeneratorRSS2 object and read in the file.

local loRSS as SFRSSGeneratorRSS2 of SFRSS.VCX, ;
  loItem as SFRSSItem of SFRSS.VCX
loRSS = newobject('SFRSSGeneratorRSS2', 'SFRSS.VCX')
loRSS.Load(filetostr('session.xml'))

* Display the channel information.

messagebox('Description: ' + loRSS.Description + chr(13) + ;
  'Link: ' + loRSS.Link + chr(13) + ;
  'Last build date: ' + transform(loRSS.LastBuildDate), ;
  0, loRSS.Title)

* Display item information.

for each loItem in loRSS.Items
  messagebox('Description: ' + loItem.Description + chr(13) + ;
    'Link: ' + loItem.Link + chr(13) + ;
    'Published: ' + transform(loItem.PublicationDate) + chr(13) + ;
    'Author: ' + loItem.Author + chr(13) + ;
    'GUID: ' + loItem.GUID, ;
    0, loItem.Title)
next loNode
```

This mechanism makes it easy to deal with attributes and automatically decodes encoded characters, but does require that you learn the XML DOM object model and XPath expressions to at least a minimal level to do the parsing. Also, unlike STRTOFILE() and STREXTRACT(), the XML DOM object can be very slow with large documents.

A third approach is to use the VFP XMLAdapter object, which collaborates with an internal XML DOM object to create a VFP cursor. I won't go into detail on this mechanism, as Craig Boyd has documented it in several articles in his blog:

http://www.sweetpotatosoftware.com/SPSBlog/PermaLink,guid,611b18e5-9e33-4082-9cd2-326005c940db.aspx
http://www.sweetpotatosoftware.com/SPSBlog/PermaLink,guid,92040996-d3bd-414c-a798-c99ddee86d3c.aspx
http://www.sweetpotatosoftware.com/SPSBlog/PermaLink,guid,9ef75db5-f3ec-4ab4-b43b-8134ef0af070.aspx

XMLAdapterParse.PRG is a slightly modified version of Craig's code that demonstrates this.

Uses for RSS in VFP Applications

Now that we know how to generate and consume RSS with VFP, what good is it? Here are several ideas I and others (including Ted Roche and Rick Borup) have come up with.

Publishing Application Errors

In his RSS white paper "Integrating RSS with Visual FoxPro Applications" (http://www.ita-software.com/papers/Borup_RSS.pdf), Rick Borup presented a cool idea: publishing application errors as RSS. The idea is that an application's error handler would generate RSS documenting an error and push it to a Web site. That way, the developer and even the end-user could review errors that occurred within the application by subscribing to the feed. When bugs are fixed, the feed can be updated so everyone can see the current status of the application.

Publishing Project Updates

Ted Roche briefly mentioned an idea in his May 2004 article in FoxTalk titled "RSS: Publishing News via XML:" having a project hook that logs when files are added to a project. All developers on a project could subscribe to the feed so they're notified about project changes. SFRSSProjectHook in SFRSSProjectHook.VCX expands upon this idea by updating an RSS file when a file is added, removed, or modified.

The Init method instantiates an SFRSSGeneratorRSS2 object into the oRSS property, sets cRSSFile to the name of the RSS file to update (the name of the current project with an XML extension), loads any existing RSS information in that file, and gets the name of the current user into the cUserName property.

```
#define cnUSER_BUFFER_SIZE  64
#define ccNULL              chr(0)
local lcName, ;
  lnSize, ;
  lcUser, ;
  lnStatus, ;
  lcUserName
with This

* Instantiate an SFRSSGeneratorRSS2 object.

  .oRSS = newobject('SFRSSGeneratorRSS2', 'SFRSS.vcx')

* The name of the RSS file to write to is the project name with an XML
* extension.

  if type('_VFP.ActiveProject.Name') = 'C'
    .cRSSFile = forceext(_VFP.ActiveProject.Name, 'xml')

* Load any existing RSS.

    .oRSS.Load(.cRSSFile)

* Set some properties if they haven't already been.

    if empty(.oRSS.Title)
      .oRSS.Title = 'Project Updates for ' + _VFP.ActiveProject.Name
    endif empty(.oRSS.Title)
  endif type('_VFP.ActiveProject.Name') = 'C'

* Get the name of the user from Windows.

  lcName = ccNULL
  lnSize = cnUSER_BUFFER_SIZE
  lcUser = replicate(lcName, lnSize)
  declare integer WNetGetUser in Win32API string@ cName, string@ cUser, ;
    integer@ nBufferSize
  lnStatus   = WNetGetUser(@lcName, @lcUser, @lnSize)
  .cUserName = iif(lnStatus = 0, ;
    upper(strtran(left(lcUser, lnSize), ccNULL)), '')
endwith
```

UpdateRSS updates the RSS file when something happens to the project.

```
lparameters tcTitle, ;
  tcDescription
local loItem
with This.oRSS
  .LastBuildDate = datetime()
  loItem = .AddItem()
  with loItem
    .Title           = tcTitle
    .Description     = tcDescription
    .PublicationDate = datetime()
  endwith
  strtofile(.GetRSS(), This.cRSSFile)
endwith
```

QueryAddFile, QueryModifyFile, and QueryRemoveFile call UpdateRSS, passing it the appropriate information to log in the RSS file. Here’s the code from QueryAddFile:

```
lparameters tcFileName
local lcTitle, ;
  lcDescription
lcTitle       = 'File added to project'
lcDescription = tcFileName + ' was added to the project by ' + This.cUserName
This.UpdateRSS(lcTitle, lcDescription)
```

To see this in action, open the Test project included in the source code files accompanying this document and add, modify, or remove files. The project hook updates Test.XML accordingly.

Publishing Table Structure Changes

When the structure of a table changes, it's helpful to publish what changes were made so all team members are informed. PublishTableChanges.PRG contains a class, also named PublishTableChanges, which does just that. I won't discuss the code in detail because it's fairly lengthy, but will describe how it works.

DBCX is a public domain data dictionary that maintains meta data for the tables in a database container or free tables. You can create your own meta data for the tables in an application by instantiating the DBCXMgr class in DBCX.VCX and calling the Validate method or using one of the several commercial products based on DBCX (including Stonefield Database Toolkit and Visual FoxExpress). PublishTableChanges expects there's an existing set of meta data for the table you want to publish changes for because it uses that as the "before" snapshot of the table's structure.

After changing the structure of a table, instantiate PublishTableChanges and set various properties, including cMetaDataPath, which contains the location of the DBCX meta data, cRSSFileName, which contains the name of the RSS file to generate, and cFormat to the RSS specification you want to use (currently only RSS 2.0 is supported). Then call the PublishTableChanges method, passing the name of the table to create RSS for. That method compares the current structure of the table to the previous structure as specified in the meta data and generates XML into the specified RSS file if there are any differences. To see how this class works, run TestTableChanges.PRG, which creates a table, generates meta data for it, alters the structure, and then uses PublishTableChanges to generate a feed for the changes.

Replacing Reporting

In a sales order system, invoices could be posted to an RSS feed as they're saved. The sales manager can subscribe to this feed and immediately see sales as they happen. If that's too much information, a day-end process could summarize sales for the day and post that instead, either as the content of an item in the feed or linked to an HTML or PDF file. This alleviates the sales manager from having to run sales reports, which can be a problem if the data isn't stored locally.

In a customer relationship management (CRM) system, appointments or call lists can be posted to an RSS feed so sales people can get their "to do" list even when they're on the road. As they complete calls, these records are also posted to an RSS feed so their managers can review what their sales people are doing. Again, this allows a manager to simply subscribe to a feed rather than having to run reports on a regular basis.

The Future of RSS

Microsoft is getting into RSS in big way, adding support for it in Windows Vista, Microsoft Office, Microsoft CRM, Internet Explorer, and so forth. They even have a section of their Web site (http://msdn.microsoft.com/xml/rss/) dedicated to RSS for developers.

Because RSS is so useful and easy to implement, various organizations are thinking of all sorts of interesting uses for it. Some of these uses will require extending the RSS specifications in some ways, and there are a number of extensions already underway. For example, Microsoft has published specifications for Simple Sharing Extensions, which allows bidirectional publishing of information in RSS feeds (http://msdn.microsoft.com/xml/rss/sse/), and Simple List Extensions, which allows things like sorting and filtering feed items (http://msdn.microsoft.com/xml/rss/sle/).

Resources

There's a ton of information available to learn more about RSS. A good place to start is to Google "RSS tutorial" and read some of the many documents that come up.

Ted Roche published a couple of articles in FoxTalk, "RSS: XML Publish and Subscribe Using Visual FoxPro" in the April 2004 issue and "RSS: Publishing News via XML" in the May 2004 issue, that I found to be great introductions to the subject. These articles and the session notes from a session he did on RSS at the 2004 DevEssentials conference are available on his Web site (http://www.tedroche.com).

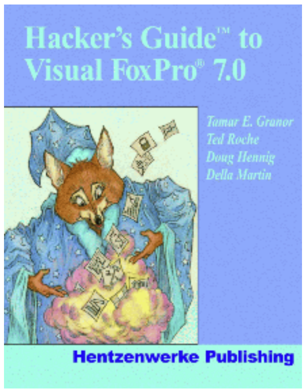
Rick Borup did a great presentation on RSS titled “Integrating RSS with Visual FoxPro Applications” at the 2005 Southwest Fox conference, and the white paper for that session is available on his Web site (http://www.ita-software.com).

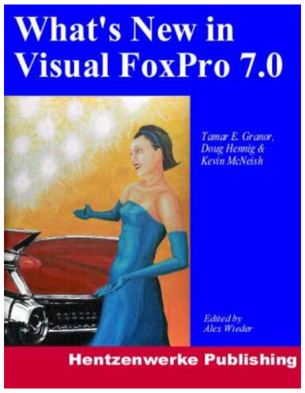
Summary

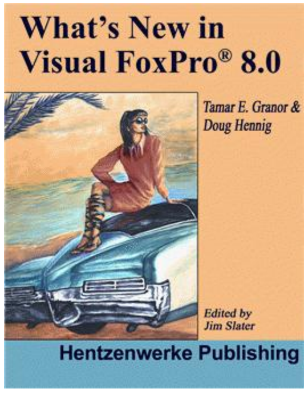
RSS is taking the world by storm! It’s an extremely useful way to publish content to subscribers using a pull mechanism. Blogs and new syndication are the most popular uses for it, but as I discussed in this document, there are many other uses for it, including those in VFP applications. Since RSS is just XML, which is just text, VFP is a great tool to both generate and consume RSS.

Biography

Doug Hennig is a partner with Stonefield Systems Group Inc. and Stonefield Software Inc. He is the author of the award-winning Stonefield Database Toolkit (SDT), the award-winning Stonefield Query, and the MemberData Editor, Anchor Editor, and CursorAdapter and DataEnvironment builders that come with Microsoft Visual FoxPro. Doug is co-author of the “What’s New in Visual FoxPro” series (the latest being “What’s New in Nine”) and “The Hacker’s Guide to Visual FoxPro 7.0.” He was the technical editor of “The Hacker’s Guide to Visual FoxPro 6.0” and “The Fundamentals.” All of these books are from Hentzenwerke Publishing (http://www.hentzenwerke.com). Doug writes the monthly “Reusable Tools” column in FoxTalk. He has spoken at every Microsoft FoxPro Developers Conference (DevCon) since 1997 and at user groups and developer conferences all over the world. He is one of the administrators for the VFPX VFP community extensions Web site (http://www.codeplex.com/Wiki/View/aspx?ProjectName=VFPX). He has been a Microsoft Most Valuable Professional (MVP) since 1996.




What's New in Visual FoxPro 7.0
Tamar E. Granor, Doug Hennig & Kevin McNeish
Edited by Alex Wieder
Hentzenwerke Publishing


What's New in Visual FoxPro® 8.0
Tamar E. Granor & Doug Hennig
Edited by Jim Slater
Hentzenwerke Publishing