This paper was originally presented at the Southwest Fox conference in Phoenix, Arizona in October, 2005.

Integrating RSS with Visual FoxPro Applications

Rick Borup
Information Technology Associates
701 Devonshire Drive, Suite 127
Champaign, IL 61820
Email: rborup@ita-software.com
Blog: http://rickborup.com/blog/

Overview

The year 2005 has seen a real explosion of interest in RSS. This can be seen in the proliferation of blogs, in the number of Web sites now offering RSS feeds, in the increasing number and variety of RSS readers, and in the adoption of RSS for uses beyond those originally envisioned by its inventors. Among the derivative uses of RSS is the concept of using it for its inherent content rather than merely as a way of summarizing and linking to other content. This concept stimulates many ideas for incorporating RSS into Visual FoxPro applications.

This paper discusses what RSS is and what it's good for, examines the structure of RSS feeds and how to validate them, explores ways to generate RSS feeds from Visual FoxPro, and presents some real-life examples of using RSS with Visual FoxPro applications.

What is RSS?

Conceptually, RSS is a way of sharing information about Web site content with others in a standardized manner. The original idea was to create a standardized way to summarize Web site content and to publish notification when that content was updated or changed. Looking at from a technical perspective, RSS is an XML-based file format for publishing metadata information about Web site content.

RSS Terminology and Standards

Before jumping much further into a discussion of RSS, it is helpful to become familiar with some of the common terminology. Sometimes there are several terms for the same concept, and sometimes one term can have different meanings. Even the term RSS itself has a dual meaning.

RSS as an acronym

As an acronym, the term RSS has had several definitions over the years. One of the early specifications for publishing metadata about Web site content was called RDF, for Resource Description Framework, and one of the original definitions of the RSS acronym was RDF Site Summary. As the specifications evolved, so did the meaning of RSS. Although sometimes used to mean Rich Site Summary, the most commonly accepted definition of the RSS acronym today is Really Simple Syndication.

RSS as a specification

The term RSS also applies to a set of formal specifications for XML-based syndication of Web site content. The RSS specification has evolved through several versions spanning 0.9, 0.9x, 1.0, 2.0, and the recently proposed 3.0. The RSS 2.0 specification is currently the most widely used of these, and to many people it defines the current *de facto* standard for Really Simple Syndication.

Atom is a newer and competing specification. Although the Atom specification remained at version 0.3 for quite a while, version 1.0 was recently published and is rapidly gaining acceptance. Some people feel there is no need for another standard, but Atom has nonetheless been adopted by several publishers, and many feed readers now support the Atom syndication format as well as the RSS format.

Feeds

Another fundamental term in the RSS world is the word *feed*. A feed, sometimes also called an RSS feed or a news feed, is simply a way of referring to a source for RSS, i.e., a Web-accessible XML file in one of the standard formats.

The term *RSS feed* is used both in the more generic sense to refer to any feed, either RSS-based or Atom-based, and also in the more specific sense to refer only to feeds that conform to one of the RSS specifications such as RSS 2.0. Because of the potential for confusion, you may hear people using other terms for the more generic meaning of RSS feed. Microsoft, for example, has recently adopted the term *Web Feed* in the context of Internet Explorer 7 beta.

Feed readers

The term *feed reader*, also known as a *news aggregator*, refers to a piece of software that knows how to access and display feeds. Feed readers come in many varieties including stand-alone PC applications, browser-based readers, solutions integrated into e-mail programs such as Microsoft® Outlook, and readers for devices other than PCs.

Items and groups of items

In the RSS specification, each block of information within a feed is called an *item*, and a group of related items in the same feed is known as a *channel*. In the Atom specification, the corresponding terms are *entry* and *feed*. Feed readers may use their own terms for these things. For example, in the latest version of FeedDemon a feed is referred to as a *subscription*.

What is RSS good for?

In general, RSS is a good choice for any situation where relatively frequent but irregular notification of new or updated information needs to be delivered via the Web. Common uses of RSS include news feeds, which typically consist of headlines with a link to the full story, Web site update notifications, which may provide a summary along with a link to the new or updated content, and of course blogs, which are a hugely popular application of RSS.

While originally intended to provide summary information with a link to full content somewhere else on the Web, people are discovering there are situations where RSS is useful for its inherent content alone. In other words, RSS can be useful even if you don't use it to link to something else. This is one area where RSS offers numerous opportunities for integration into other applications, and is the basis for the real-life examples of integrating RSS into Visual FoxPro applications described later in this paper.

Structure of an RSS file

Conceptually, an RSS file contains two primary types of information:

1. Information about the feed itself; this typically includes a title and a description, a link to the full content (usually a Web page), the date the feed was last updated, and other information.
2. Information about each of the articles, or items, within the feed; this includes a title, a description of the content (summary or full), the date the item was published, a link to the full content (if any), and other information.

The syntactical structure of an RSS file is governed by the specification you've chosen to follow. Although there can be considerable difference between the syntax of a feed published to one specification versus another specification, keeping the conceptual structure in mind should help you understand the syntactical structure of any specification. In this paper I'm using the RSS 2.0 specification for all examples.

Structure of an RSS 2.0 XML file

The outermost element in an RSS 2.0 feed, not counting the XML declaration itself, is the RSS version declaration, which is written as

```
<rss version="2.0">
```

This is followed immediately by the channel element opening tag, which in turn is followed by the channel sub-elements. Within an RSS 2.0 channel element, the title, link, and description sub-elements are required, while other sub-elements are optional. As with all XML, closing tags are always required. In the simplest case, then, a channel element would look something like this:

```
<channel>
   <title>My RSS Feed</title>
   <link>http://myRSSFeed.com</link>
   <description>My RSS Feed</description>
   . . . item(s) go here . . .
</channel>
```

The item element(s) are nested within the channel element. Each item element must have a title or a description sub-element, and may have other optional sub-elements. A simple item element might look like this:

```
   <item>
      <title>My first RSS feed</title>
      <description>This is my first RSS feed.</description>
      <link>http://myRSSFeed.com/article1.htm</link>
   </item>
```

To complete the RSS file, embed the item element within the channel element and add the closing tag for the outermost RSS element. You now you have a complete RSS 2.0 file, as shown in Listing 1.

Listing 1: A simple RSS 2.0 file

```
<rss version="2.0">
<channel>
   <title>My RSS Feed</title>
   <link>http://myRSSFeed.com</link>
   <description>My RSS Feed</description>
   <item>
      <title>My first RSS feed</title>
      <description>This is my first RSS feed.</description>
      <link>http://myRSSFeed.com/article1.htm</link>
   </item>
</channel>
</rss>
```

The code in Listing 1 is available as example1.rss in the session downloads.

Validation

If you want to conform to standards, it's important to create valid RSS. Actually there are two levels of validity to be concerned with: the XML itself, and the conformity of the XML to the desired RSS specification.

XML validation

In the sense we're concerned about it here, valid XML means following the rules for writing well-formed XML. The rules are:

1. there is a single root element
2. elements are properly nested
3. all elements have closing tags
4. all attributes are quoted

As always, remember that XML is case sensitive; for example, *Channel* is not the same as *channel*. In addition, use entity encoding for special characters when they occur within the content of an element; for example, the left and right angle brackets should be written as *<* and *>*, respectively.

RSS validation

After ensuring your XML is well-formed, you'll also want to want to ensure your RSS is valid by checking that it conforms to the desired RSS specification. Probably the easiest way to do this is to run your file through the free online validation service called FEED Validator, which is available at http://feedvalidator.org.[1]

FEED Validator validates your feed against the specification declared in the feed file. FEED Validator supports both RSS 2.0 and Atom 1.0. If there are any validation errors FEED Validator tells you about them in sufficient detail for you to locate and resolve the error(s). If there are no errors it tells you your feed is valid. I've found this service to be quite useful in detecting errors that otherwise might have gone unnoticed in my own feeds.

Why care about RSS validity?

Why should you care about producing valid RSS? On the one hand, if you're designing strictly for private use—in other words, if you're working with a private feed where you control both the generating side and the consuming side of the feed and nobody besides you or your own apps will be consuming it—then you may not have to care much about RSS validation. After all, it's only XML, and you can make XML work however you want it to.

On the other hand, if you don't stick to RSS standards then feed readers and other apps (perhaps even your own) that expect standards-compliant RSS may not be able to properly consume your feed. Besides, if you don't stick to one of the standards you're not really doing RSS. So my advice is, be sure your RSS is valid; it's not hard to do and it'll make things easier for you in the long run.

---

[1] Validating a feed with FEED Validator requires that your feed file have a publicly accessible URL. You can't use FEED Validator to validate an RSS file on your local machine unless you're running a Web server on that machine and can supply a public URL that points to the local file.

A second example

Now that we've talked about XML and RSS validation, let's take a look at a second example of an RSS 2.0 file. This one includes a few more sub-elements and is intended to be a more realistic example of a file you'd actually publish. The code in Listing 2 is available as example2.xml in the session downloads.

Listing 2: A more complete RSS 2.0 file

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0">
<channel>
   <title>My RSS Feed</title>
   <link>http://myRSSFeed.com</link>
   <description>My RSS Feed</description>
   <lastBuildDate>Sun, 11 Sep 2005 18:08:09 GMT</lastBuildDate>
   <item>
      <title>My first RSS feed</title>
      <description>This is my first RSS feed.</description>
      <link>http://myRSSFeed.com/article1.htm</link>
      <author>rborup@ita-software.com (Rick Borup)</author>
      <guid isPermaLink="false">SWFox2005-123abc-Foo-999-2</guid>
      <pubDate> Sun, 11 Sep 2005 18:08:09 GMT </pubDate>
   </item>
</channel>
</rss>
```

There are several differences between this file and the first example. For starters, this file begins with an XML declaration, which includes a character encoding specification. Although your choice of character encoding may vary, you should always begin your RSS file with the XML declaration.

Within the channel element, notice the lastBuildDate sub-element below the description. This optional sub-element tells consumers of the feed when the feed was last updated. As illustrated in Listing 2, the RSS specification requires the date and time to be in RFC-822 format.[2]

Listing 2 also introduces three new sub-elements within the item element. The author element, as you would expect, identifies the author of the item. By the way, this is a good example of where feed validation helped me. The first time I wrote an RSS 2.0 feed I put my name in the author tag. This seemed logical to me, but FEED Validator flagged it as an error. I discovered I hadn't read the RSS 2.0 specification carefully enough, because the author tag requires an e-mail address rather than just a name.

The optional guid tag is used to uniquely identify an item among all other items. In the RSS 2.0 specification, the guid can take one of two forms: it can be a link to the full content the item refers to, or it can be a simple string. If it's a string, it can be an actual GUID (in the Microsoft® sense of the word) or it can simply be any unique string of characters. If you use a string instead of a

[2] It's fairly easy to write a method to turn a VFP datetime variable into an RFC-822 string, but you don't have to do that yourself because it's already been done for you. In an article he wrote for the May, 2004 edition of FoxTalk 2.0, Ted Roche included a program named RFC822Date.prg. This nifty little program not only converts a VFP datetime variable to RFC-822, but also reads your machine's time zone setting and converts your local time to GMT. As with all of Ted's code, it's a very nice piece of work.

link, you need to include the *isPermalLink* attribute with a value of *false* in order to pass validation. An item's guid should never change, regardless of whether it's a URL or a string.

The pubDate element specifies the item's publication date and time. Like the lastBuildDate, the pubDate must also be in RFC-822 format, but unlike the lastBuildDate, a pubDate element can have a value in the future. According to the specification, feed readers aren't supposed to display an item until its pubDate becomes current; this allows a publisher to physically post an item to a feed file in advance of the time it should become publicly available. Not all feed readers respect this standard, though, so the pubDate is commonly used for the actual date and time the item was published. By the way, the channel element can have a pubDate, too.

Generating RSS from VFP

Okay, you've been reading (or maybe skipping ahead) for about six pages now and you're thinking, "finally he's getting around to what I really want to know about!" Yes, this is where the VFP rubber meets the RSS road. Let's see if we can do something useful here without skidding out of control.

Creating the feed

Creating an RSS feed from within a VFP app is pretty straightforward. An RSS file, after all, is simply a specific kind of XML file. XML is string-based, and VFP excels at creating and manipulating strings. So while there are other ways to generate XML from VFP, string manipulation would seem to be a good choice.

The *RSSHandler* class

To illustrate this, I wrote a VFP class named RSSHandler to create an XML file in RSS 2.0 format. This class handles the process in four steps, and each step has a corresponding method in the class. These methods are discussed below. The entire class is available in the session downloads as myRSSHandler.PRG.

Caveat emptor

Because it is demo code, the RSSHandler class does not do many of the things a full-blown class should do. This includes but is not limited to the following:

- None of the methods do any kind of parameter validation, so you can trigger errors by passing them something other than string values at runtime.
- Although the class generates the channel and item sub-elements required by the RSS 2.0 specification, it does not check that the parameter values you pass for those sub-elements are valid. In other words, you can pass the empty string or you can pass a date-time value that is not RFC-822 format and the class will not tell you about it. Doing this type of thing won't trigger an error but it may result in invalid RSS.
- The class does not enable you to create any of the other optional sub-elements that are allowed by the RSS 2.0 specification.

The class uses STRTOFILE to create the actual XML file. Because STRTOFILE overwrites existing files without warning if SET SAFETY is OFF, the RSSHandler class *does* check to see if there is an existing file of the same name, and if it finds one it warns you before overwriting it. Nonetheless, you should use care and common sense when specifying the file name and location for your XML output file; if you override the warning, nothing prevents you from overwriting any file on your machine with the XML file generated by this class.

The *InitializeFeed* method

The RSSHandler class creates the XML as a string named cXML, which is a property of the class. The *InitializeFeed* method starts a new cXML string and populates it with the static XML common to all RSS 2.0 feeds along with the specific channel information you pass to this method in its four parameters. Here is the *InitializeFeed* method code:

```
*-----------------------------------------------------------------
*   RSSHandler :: InitializeFeed
*---------------------------------
*   Function: Generate the XML to initialize a feed and open its
*             channel element.
*       Pass: tcTitle         - channel title
*             tcLink          - channel link
*             tcDescription   - channel description
*             tcLastBuilddate - channel lastBuildDate (RFC 822 format)
*     Return: Nothing
*   Comments:
*-----------------------------------------------------------------
PROCEDURE InitializeFeed( tcTitle as String, ;
                          tcLink as String, ;
                          tcDescription as String, ;
                          tcLastBuildDate as String) as VOID
TEXT TO this.cXML TEXTMERGE NOSHOW
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0">
<channel>
   <title><<tcTitle>></title>
   <link><<tcLink>></link>
   <description><<tcDescription>></description>
   <lastBuildDate><<tcLastBuildDate>></lastBuildDate>

ENDTEXT && Keep the extra line feed before ENDTEXT.
ENDPROC && InitializeFeed
```

The blank line before ENDTEXT is there to make the generated code look better when you open it in an editor.

The *InsertItem* method

The *InsertItem* method inserts an item into the XML string. Because a channel can have many items, you can call this method as often as necessary, once for each item you want to include. This method accepts six parameters representing the six sub-elements the method is capable of producing.

```
*-----------------------------------------------------------------
*   RSSHandler :: InsertItem
*---------------------------------
*   Function: Generate the XML to add an item to the feed.
*       Pass: tcTitle       - item title
*             tcDescription - item description
```

```
*             tcLink        - item link
*             tcAuthor      - item author (e-mail address)
*             tcGUID        - item guid
*             tcPubDate     - item pubDate (RFC 822 format)
*     Return: Nothing
*   Comments:
*-----------------------------------------------------------------
PROCEDURE InsertItem( tcTitle as String, ;
                      tcDescription as String, ;
                      tcLink as String, ;
                      tcAuthor as String, ;
                      tcGUID as String, ;
                      tcPubDate as String) as VOID
TEXT TO this.cXML ADDITIVE TEXTMERGE NOSHOW
   <item>
      <title><<tcTitle>></title>
      <description><<tcDescription>></description>
      <link><<tcLink>></link>
      <author><<tcAuthor>></author>
      <guid isPermaLink="false"><<tcGUID>></guid>
      <pubDate><<tcPubDate>></pubDate>
   </item>

ENDTEXT && Keep the extra line feed before ENDTEXT.
ENDPROC && InsertItem
```

As with the *InitializeFeed* method, the blank line before ENDTEXT is there to make the generated code look better when you open it in an editor.

The *FinalizeFeed* method

The FinalizeFeed method simply closes the channel and rss tags. It should be called after the last *InsertItem* method call.

```
*-----------------------------------------------------------------
*   RSSHandler :: TerminateFeed
*---------------------------------
*   Function: Generate the XML to terminate the feed.
*       Pass: Nothing
*     Return: Nothing
*   Comments:
*-----------------------------------------------------------------
PROCEDURE TerminateFeed() as VOID
TEXT TO this.cXML ADDITIVE NOSHOW
</channel>
</rss>
ENDTEXT
ENDPROC && TerminateFeed
```

The *WriteFile* method

The WriteFile method uses STRTOFILE to write the contents of the cXML string, which was built by the other methods, to the file whose name you specify in the parameter.

```
*-----------------------------------------------------------------
*   RSSHandler :: WriteFile
*---------------------------------
*   Function: Write the XML string to a file.
*       Pass: tcFileName - the name of the XML file to be written.
*     Return: Numeric - the number of bytes written
*   Comments:
*-----------------------------------------------------------------
PROCEDURE WriteFile( tcFileName as String) as Integer
IF FILE( tcFileName)
```

```
   IF MESSAGEBOX( "File " + ALLTRIM( tcFileName) + " already exists." + ;
                  CHR(13) + CHR(13) + ;
                  "Do you want to overwrite it?", ;
                  MB_YESNO + MB_ICONQUESTION, "Create RSS File") <> IDYES
      RETURN 0
   ENDIF
ENDIF
RETURN STRTOFILE( this.cXML, tcFileName)
ENDPROC && WriteFile
```

Using the RSSHandler class

To use the RSSHandler class, instantiate it in the VFP IDE or in a program and call its four methods in the correct order. To demonstrate this, I've written a little test program called GenerateRSSFile1.PRG, which is available in the session downloads.[3]

In this example I'm using literals as the source of information for the feed. In a real-life situation, of course, you would most likely be pulling this information from a VFP table or other data source. You can easily modify this test program to suit your own needs and/or to make it work with data instead of literals.

Listing 3: GenerateRSSFile1.PRG is a test program for the RSSHandler class.

```
SET PROCEDURE TO RFC822Date
ox = NEWOBJECT( "RSSHandler", "myRSSHandler.prg")
ox.Initializefeed( ;
  "My RSS Feed", ;
  "http://myRSSFeed.com", ;
  "My RSS Feed", ;
  RFC822Date( DATETIME()) ;
)
ox.InsertItem( ;
  "My First RSS Feed", ;
  "This is my first RSS feed.", ;
  "http://myRSSFeed.com/article1.htm", ;
  "rborup@ita-software.com (Rick Borup)", ;
  "SWFox2005-123abc-Foo-999-2", ;
  RFC822Date( DATETIME()) ;
)
ox.TerminateFeed()
ox.WriteFile( "myRSSFile1.xml")
```

This program generates an XML file very similar to the one shown in Listing 2 and writes it to disk as myRSSFile1.xml, an arbitrary file name chosen for this example. To view the resulting file, simply do a *MODI FILE myRSSFile1.xml* in the VFP command window or open it in an XML-aware browser such as Microsoft Internet Explorer.

Although the information it contains is useless except for demonstration purposes, this is a fully valid RSS 2.0 feed suitable for publication. Feel free to copy myRSSFile1.xml to your own Web site and run it through FEED Validator. Also try subscribing to it using your favorite RSS 2.0-enabled feed reader. Or, if you prefer, you can validate and/or subscribe to it using the copy I've placed on my own Web site at www.ita-software.com/SWFox2005/RSS/myRSSFile1.xml. Below

[3] GenerateRSSFile1.PRG makes a call to the previously referenced RFC822Date.PRG written by Ted Roche and available to FoxTalk 2.0 subscribers in the May, 2004 downloads. If you are unable to download that program, an alternative is publicly available at fox.wikis.com/wc.dll?Wiki~XMLDateTime.

is a screenshot of this feed as it appears in FeedDemon, which happens to be my own personal favorite feed reader.

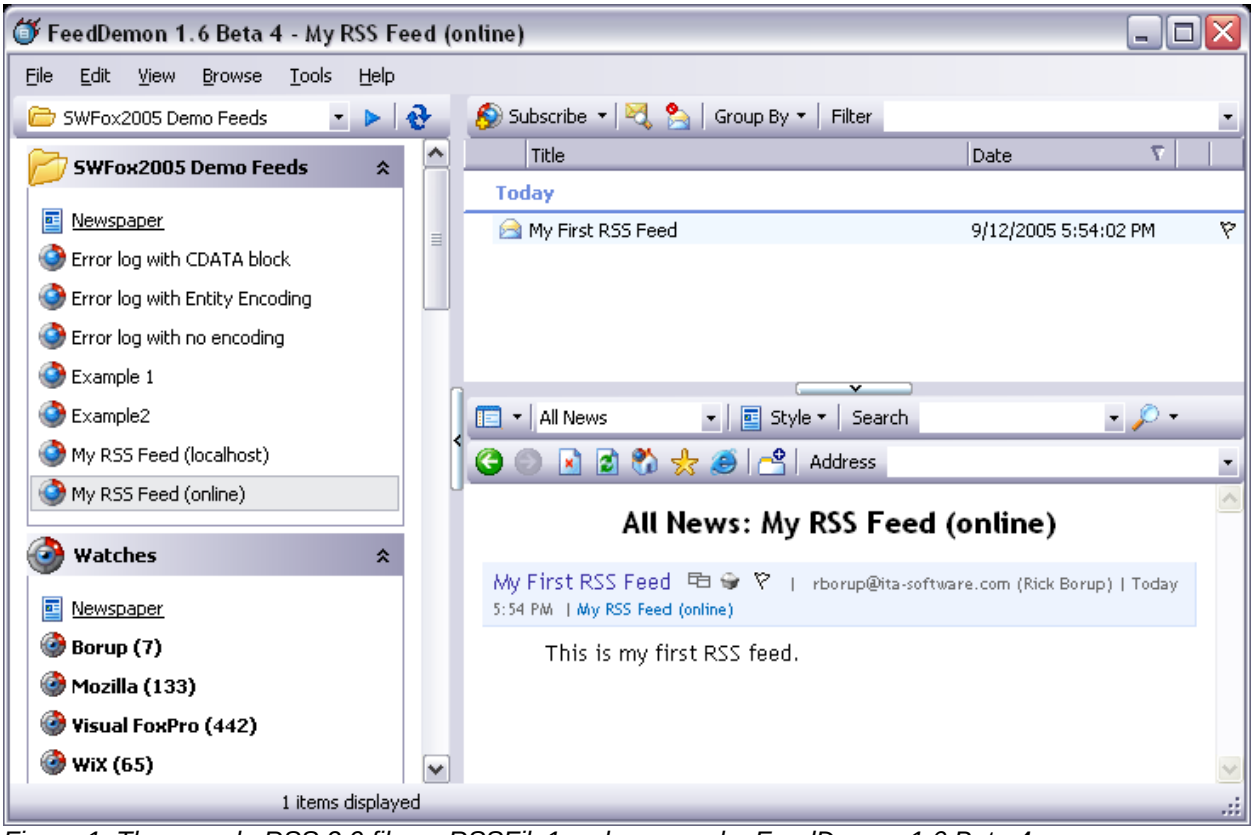

Figure 1: The sample RSS 2.0 file myRSSFile1.xml as seen by FeedDemon 1.6 Beta 4.

In Figure 1, the feed is highlighted in the upper left pane. Within FeedDemon, I've named this feed "My RSS Feed (online)" to distinguish it from "My RSS Feed (localhost)", which is the same feed served from my local IIS. The upper right pane is a list of the items in the selected feed—there's only one, of course, since that's all we generated—and the content of the selected item is shown in the lower right pane.

Note: If you view this sample feed in a feed reader that renders active links, as FeedDemon does in the lower right pane in Figure 1, I suggest you do not click on the "My First RSS Feed" or "My RSS Feed (online)" links. They point to a fictitious Web site named myRSSFeed.com, and if you click on them you'll probably be redirected to a site that will try to sell you that domain name.

Updating the feed

Updating an existing feed is essentially the same as creating it in the first place, although there are at least a couple of different ways to approach the update process. One way is to re-generate the entire XML file each time you do an update, adding new item(s) as necessary. Another way is to insert new items into the existing file. There are pros and cons to each approach.

One argument against re-generating the entire file for each update is that you may have to take special steps to preserve the pubDate and guid sub-element values of existing feed items. If these values are generated by your RSS-creation code or are input manually, as they are in my sample RSSHandler class and its test program, you could end up with different values for these elements each time you re-generated the file.

To avoid this problem you'd either have to read and parse the original XML file or you'd have to store the original values of these elements somewhere else, such as in a VFP table, and pull them from there each time the item was re-generated. On the plus side for this approach is that re-generating the entire feed file gives you the chance to clean up the file by deleting (and possibly archiving) stale or outdated content.

Another approach to updating a feed file is simply to insert or append the new item(s) into the existing file. This isn't quite as easy as re-creating the entire file because it involves parsing the existing file, but it isn't hard to see how you could modify the code in the RSSHandler class to implement this approach.

In my opinion, neither approach is always better than the other. Use whichever one works best for you in each particular situation.

Real-life examples of using RSS in VFP apps

Following are two real-life examples of using RSS in VFP apps. In both cases, a VFP app generates an RSS feed for the purpose of providing notification to interested parties—in this case, to me as the developer and/or to my clients as the users of the app—of events that occur on an irregular but not infrequent basis as the app runs on a daily basis.

Publishing an application's error log

The scenario in this case is a multi-user VFP app running on a local area network in an office environment with about a dozen users. The error handling portion of this app builds an error log in the form of a VFP table, from which I as the developer can pull reports on a periodic basis to see if errors are occurring that I'm not otherwise being informed of. (I was surprised to find out how willing some users are to ignore a non-critical error or even to cancel and restart an app without telling me something had happened!).

I figured using an error log table would be a good way to find and fix problems in my app even if the users didn't tell me about them, but it turned out to be problematic for me to get access to the error log table. I was not at the client's site very often, the client's IT staff was unwilling to provide me remote access to their file server, and trying to get their office staff to e-mail me a copy of the error log table on a regular basis didn't work very well either.

So I began to look for a different solution, and RSS seemed to be a perfect choice. What if I were able to create a feed for the error log and update it each time a new error occurred? I could use the description field for the entire block of error information. No link to any other content would be required. Then, I would subscribe to this feed from my feed reader and be notified in my office whenever an error occurred on the client's site. The feed item would include full information about the error, so I would be able to start debugging right away.

One initial concern was how to make the RSS feed available over the Web so I could read it from my office. Fortunately, the client in this case has an in-house Web server, and a portion of the Web server's public space is mapped to the local area network. This enabled my app to write the RSS feed file to a location on the Web server. Naturally I didn't want this to be a public feed, so the feed file is located in a directory that's excluded from search engine crawlers and other bots while still being accessible to anyone who knows the URL.

The error log table in this application is structured like this:

```
tDateTime  T        && date and time error occurred
cUser      C(10)    && username
nError     I        && error number
cMethod    M        && method where error occurred
nLine      I        && line number
cErrorMsg  M        && the error message
cAError3   C(50)    && the information from AERROR(3)
nAError4   I        && the information from AERROR(4)
nAError5   I        && the information from AERROR(5)
```

The code used to generate the RSS feed from this table is substantially similar to the sample RSSHandler presented in this session. In the real app there is also a method that formats the information from each record of the error log table into a string. This string is then used as the value of the description element within the feed item.

The method that creates the string for the description does several things to format the information for better viewing in the feed reader. In order to display each column's data on a separate line, for example, the formatting method inserts an HTML line-break
 tag at the end of each line. The string from the sample error message used in this session looks like this:

```
User: RICK      <br />
Error: 12<br />
Method: PROCEDURE RESULTSPROCESSOR.GETDATA CLSRESULTSPROCESSOR.FXP<br />
Line: 593<br />
Error Msg: Error 12 - Variable 'GENERAL' is not found.
Line 593 of method PROCEDURE RESULTSPROCESSOR.GETDATA CLSRESULTSPROCESSOR.FXP
Source code: SET COLLATE TO &lcCollate<br />
aError[3]: GENERAL<br />
aError[4]: 0<br />
aError[5]: 0
```

A couple of issues become apparent from this example. One issue is that the inclusion of HTML tags within the description element of an item can cause problems with the parsing of the feed. The other issue is that the presence of un-escaped special characters within the body of the description—such as the ampersand in the source code line SET COLLATE TO &lcCollate—can cause similar problems.

These problems can be avoided in one of two ways: by using entity encoding within the content of the description element, or by wrapping the entire content of the description element in a CDATA block. Entity encoding means replacing raw control characters such as < and > with their escaped equivalents < and >. The other solution is to place the entire content of the description element within a CDATA block, which is probably easier to implement because you then don't need to encode each problematic character individually. Either approach is acceptable.

Listing 4 shows the description element of the sample item without either entity encoding or a CDATA block.

Listing 4: The description element without entity encoding or a CDATA block

```
<description>User: RICK      <br />
Error: 12<br />
Method: PROCEDURE RESULTSPROCESSOR.GETDATA CLSRESULTSPROCESSOR.FXP<br />
Line: 593<br />
Error Msg: Error 12 - Variable 'GENERAL' is not found.
Line 593 of method PROCEDURE RESULTSPROCESSOR.GETDATA CLSRESULTSPROCESSOR.FXP
Source code: SET COLLATE TO &lcCollate<br />
aError[3]: GENERAL<br />
aError[4]: 0<br />
aError[5]: 0</description>
```

When viewed in the FeedDemon feed reader, the feed from Listing 4 is rendered as shown in Figure 2. Other feed readers might not render it the same way, or might not even render it at all due to the character encoding issues in the feed item.

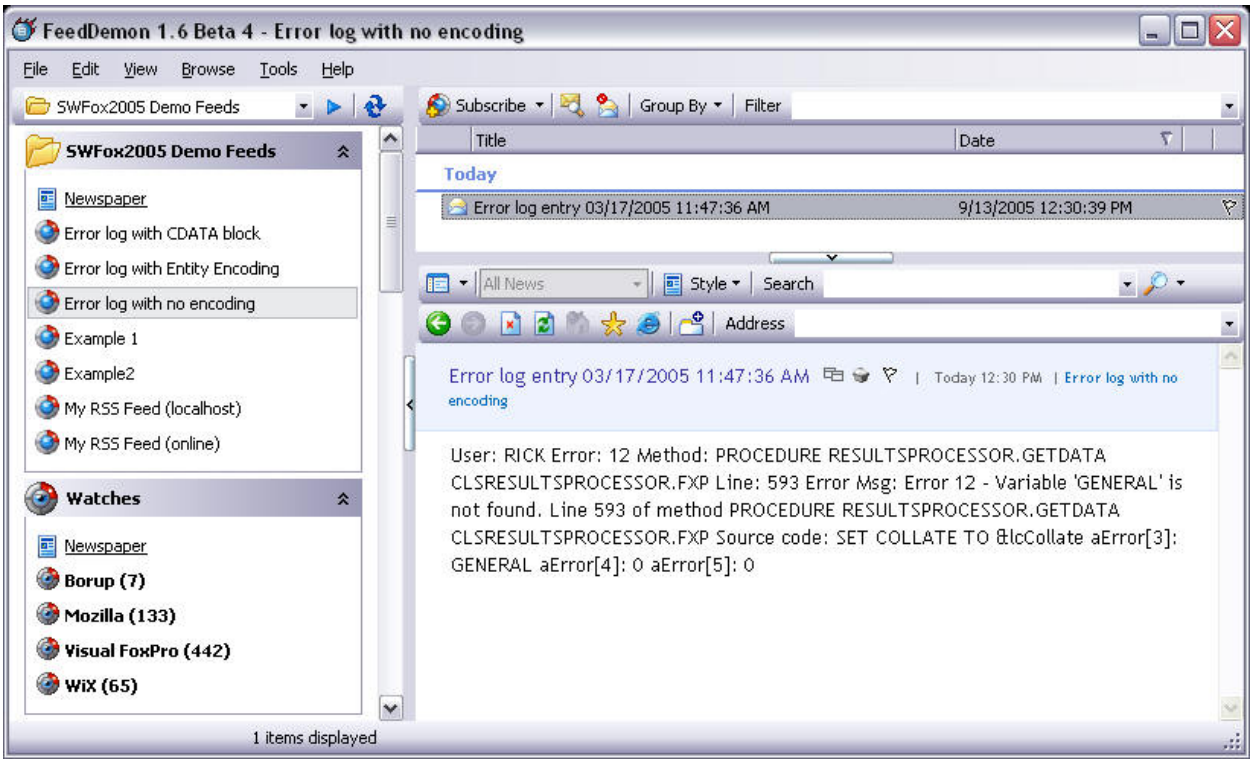

Figure 2: With no entity encoding or CDATA block, the sample error log feed is run together without line breaks.

For comparison, Listing 5 shows the description element with entity encoding while Listing 6 shows it enclosed within a CDATA block.

Listing 5: The description element with entity encoding

```
<description>User: RICK      &lt;br /&gt;
Error: 12&lt;br /&gt;
Method: PROCEDURE RESULTSPROCESSOR.GETDATA CLSRESULTSPROCESSOR.FXP&lt;br /&gt;
```

```
Line: 593&lt;br /&gt;
Error Msg: Error 12 - Variable 'GENERAL' is not found.
Line 593 of method PROCEDURE RESULTSPROCESSOR.GETDATA CLSRESULTSPROCESSOR.FXP
Source code: SET COLLATE TO &amp;lcCollate&lt;br /&gt;
aError[3]: GENERAL&lt;br /&gt;
aError[4]: 0&lt;br /&gt;
aError[5]: 0</description>
```

Listing 6: Using a CDATA block eliminates the need for entity encoding.

```
<description><![CDATA[User: RICK      <br />
Error: 12<br />
Method: PROCEDURE RESULTSPROCESSOR.GETDATA CLSRESULTSPROCESSOR.FXP<br />
Line: 593<br />
Error Msg: Error 12 - Variable 'GENERAL' is not found.
Line 593 of method PROCEDURE RESULTSPROCESSOR.GETDATA CLSRESULTSPROCESSOR.FXP
Source code: SET COLLATE TO &lcCollate<br />
aError[3]: GENERAL<br />
aError[4]: 0<br />
aError[5]: 0]]></description>
```

When viewed in FeedDemon, the item is now formatted the way I want it (see Figure 3).

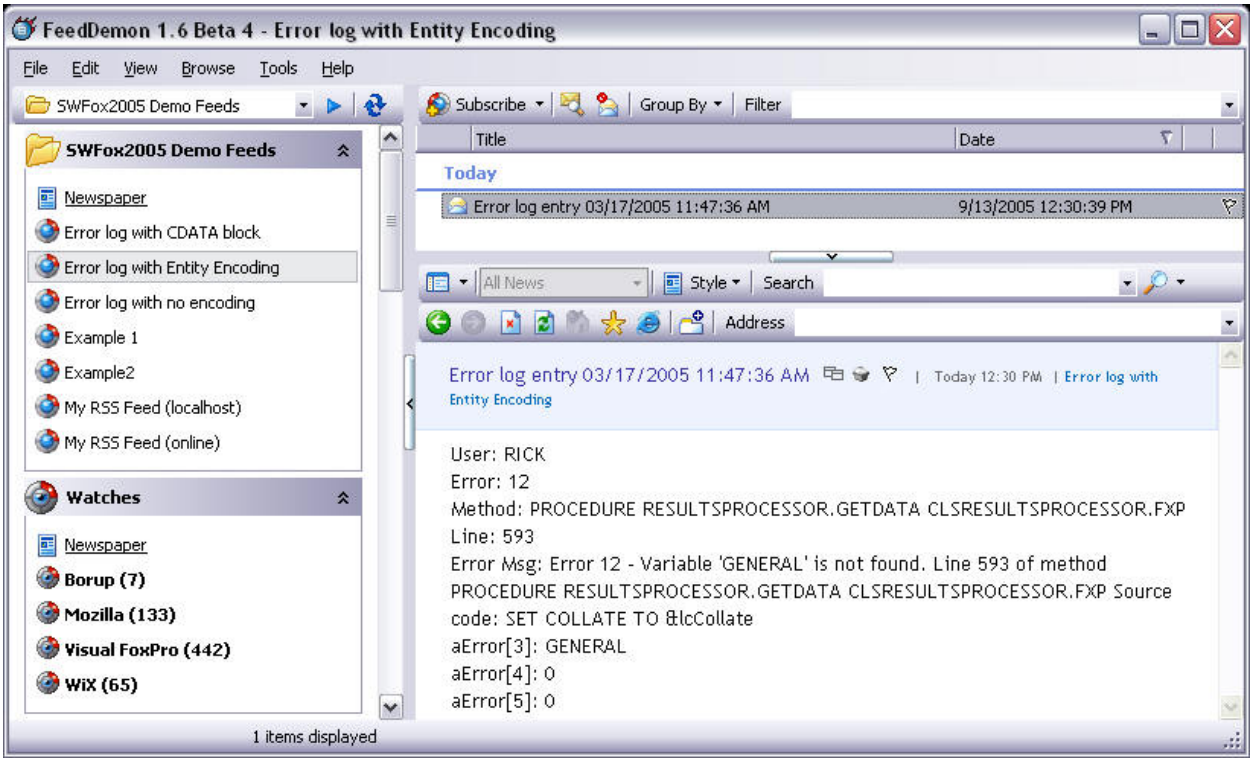

Figure 3: With entity encoding, the sample error log feed is nicely formatted and easily readable. Using a CDATA block produces the same result.

Listing 7 is some sample code to generate the error log feed using the RSSHandler class. The code is listing 7 is available as GenerateErrorLogFeed.PRG in the session downloads.

Listing 7: This code illustrates how to generate the error log feed using the RSSHandler class.

```
SET PROCEDURE TO RFC822Date, GetNewGUID ADDITIVE
ox = NEWOBJECT( "RSSHandler", "myRSSHandler.prg")
ox.Initializefeed( ;
```

```
  "My Error Log", ;
  "http://localhost/SWFox2005/myErrorLog.xml", ;
  "My Sample Error Log RSS Feed", ;
  RFC822Date( DATETIME()) ;
)
IF NOT USED( "errorLog")
  USE errorLog IN 0
ENDIF
SELECT errorLog
SCAN
  ox.InsertItem( ;
        "Error log entry" + TTOC( errorLog.tDateTime), ;
        "<![CDATA[" + ;
            "User: " + ALLTRIM( errorLog.cUser) + "<br />" + CHR(13) + ;
            "Error: " + TRANSFORM( errorLog.nError) + "<br />" + CHR(13) + ;
            "Method: " + ALLTRIM( errorLog.cMethod) + "<br />" + CHR(13) + ;
            "Line: " + TRANSFORM( errorLog.nLine) + "<br />" + CHR(13) + ;
            "Message: " + ALLTRIM( errorLog.cErrorMsg) + "<br />" + CHR(13) + ;
            "aError[3]: " + ALLTRIM( errorLog.cAError3) + "<br />" + CHR(13) + ;
            "aError[4]: " + TRANSFORM( errorLog.nAError4) + "<br />" + CHR(13) + ;
            "aError[5]: " + TRANSFORM( errorLog.nAError5) + ;
        "]]>", ;
        "http://localhost/SWFox2005/myErrorLog.xml", ;
        "rborup@ita-software.com (Rick Borup)", ;
        SUBSTR( GetNewGUID(), 2, 36), ;
        RFC822Date( DATETIME()) ;
  )
ENDSCAN
USE IN SELECT( "ErrorLog")
ox.TerminateFeed()
ox.WriteFile( "myErrorLog.xml")
```

And there you have it. Creating an RSS feed from the error log of this app has made it much easier for me as the developer to be informed on a timely basis when my app generates an error while running at the client's site.

Publishing notification of Web site database updates

The second example of integrating RSS into a VFP app involves a Web app that accesses many different databases, all of which are located on the Web server or its local resources. There are several remote users who each maintain and update their own local copy of one or more of these databases. The remote users are authorized to send updated copies of their databases to the server whenever necessary. These users want to receive confirmation when an update they have sent has been received and processed by the server.

There are two people who want to be informed whenever *any* of the databases are updated: this includes me, as the developer and site administrator, and my client, who oversees the work of all the remote users. So the two of us should receive all update notifications, while each remote user should receive notification only when one of his or her own databases has been updated.

Originally, the update notifications were handled by having the server program send an e-mail. This actually worked pretty well, but as the amount of e-mail—and spam in particular—that everybody receives on a daily basis increased, so too did the likelihood that a notification e-mail from our server might be intercepted and discarded by an overly aggressive corporate spam filter or that it might simply go unnoticed in the e-mail blizzard even if it did get through.

Different solutions were considered, and once again RSS quickly jumped to the top of the list. The server could easily publish a feed that users could subscribe to. Moreover, we would no

longer have to keep up with an ever-changing list of e-mail addresses because a feed, once established, has a fixed URL and can be accessed by anyone we give the URL to.

The mechanics of generating the RSS feed in this case are substantially the same as in the first example, so I won't go into the details again here. One difference in this case as compared to the first example, however, is that the VFP app generates more than one feed: there is a master feed that gets updated for all database updates, and there are individualized feeds for each remote user that get updated only when the server processes an update to a database maintained by that user. Each user subscribes only to his or her own feed, and the feed URL remains constant for each user.

Another advantage of using RSS over e-mail in this situation is that feed readers typically maintain a local cache of the items they've read, so these items persist on the user's machine even after they've been removed from the originating feed file. This provides the users with a built-in history of the updates they've submitted, which was a practical impossibility under the old e-mail notification system unless the user took steps to separate and store the notification e-mails somewhere other than their Inbox.

This is just one other example of a case where RSS again proved to be a viable solution and was easily integrated into an existing VFP app.

Conclusion

RSS is a hot topic these days and sometimes appears to have almost limitless potential. Certainly it has potential for many uses beyond the simple syndication of Web site content. In this paper, you've learned how to construct an RSS feed, explored ways to generate one from a VFP app, and seen a couple of examples of how the integration of RSS into VFP apps has provided real-life solutions. I hope this paper has also stimulated some thought about how you can employ RSS in your own work.

About the author

Rick Borup is an independent developer specializing in the design, development, and support of mission-critical business software solutions for small to medium-size businesses. Rick earned B.S. and M.B.A. degrees from the University of Illinois at Urbana-Champaign, and is owner and president of Information Technology Associates in Champaign, Illinois. He has been developing solutions with FoxPro/Visual FoxPro (VFP) full-time since 1993, and is a Microsoft Certified Solution Developer (MCSD) and a Microsoft Certified Professional (MCP) in VFP.