

*Developers Guide to Syndicating News and Blogs*



*Developing Feeds with*

# RSS *and* Atom

O'REILLY®

*Ben Hammersley*

*A facility for quotation covers the absence of original thought.*

—Dorothy L. Sayers, Lord Peter Wimsey in *Gaudy Night*

This chapter describes the RSS 2.0 specification in detail, how it works, and how it is created. It also explores RSS 2.0 predecessors—the largely compatible 0.91 and 0.92 specifications—and how they relate and can be converted to the latest standard.

## Bringing Things Up to Date

RSS 2.0 has a long history. As was shown in Chapter 1, it's based on a succession of specifications: RSS 0.91, 0.92, 0.93, and 0.94. Because of this history and because of a lack of any adequate documentation for many of these standards, there is a massive gulf between the quality of the document you can produce and the quality of what you might have to parse. In other words, many people are doing it wrong.

This confusion forces this chapter to address two different issues. The first is how to create a perfectly specification-compliant feed, and the second is how to deal with feeds produced by those with less exacting standards.

This decision brings us to another one: what to do about the older versions that led to 2.0? The answer is this: although many people are still learning to produce 0.91, 0.91, et al, we will not. You'll learn how to parse them, but from now on, as far as the simple strain of syndication feeds goes, we'll be creating only 2.0 feeds.

With that decided, steel yourself, visit the official specification document for RSS 2.0 at <http://blogs.law.harvard.edu/tech/rss>, and let's get on with it.

# The Basic Structure

The top level of an RSS 2.0 document is the `rss version="2.0"` element. This is followed by a single `channel` element. The `channel` element contains the entire feed contents and all associated metadata.

## Required Channel Subelements

There are 3 required and 16 optional subelements of `channel` within RSS 2.0. Here are the required subelements:

### title

The name of the feed. In most cases, this is the same name as the associated web site or service.

```
<title>RSS and Atom</title>
```

### link

A URL pointing to the associated resource, usually a web site. The link must be an IANA-registered URI scheme, such as `http://`, `https://`, `news://`, or `ftp://`, though it isn't necessary for an application developer to support all these by default. The most common by a large margin is `http://`. For example:

```
<link>http://www.benhammersley.com</link>
```

### description

Some words to describe your channel.

```
<description>This is a nice RSS 2.0 feed of an even nicer weblog</description>
```

Although it isn't explicitly stated in the specification, it is highly recommended that you *do not* put anything other than plain text in the `channel/title` or `channel/description` elements. There are some existing feeds with HTML within those elements, but these cause a considerable amount of wailing, and at least a small amount of gnashing of teeth. *Do not do it*. Use plain text only in these elements. The following sidebar, "Including HTML Within title or description," gives a fuller account of this, but in my opinion it's a bad idea.

## Optional Channel Subelements

There are 16 optional `channel` subelements of RSS 2.0. Technically speaking, you can leave these out altogether. However, I encourage you to add as many as you can. Much of this stuff is static; the content of the element never changes. Placing it into your RSS template or adding another line to a script is little work for the additional value of your feed's metadata. This is especially true for the first three subelements listed here:

## Including HTML Within title or description

Since the early days of RSS 0.91, there's been an ongoing debate about whether the `item/title` or `item/description` elements may, or should, contain HTML. In my opinion, they should not, for both practical and philosophical reasons. Practically speaking, including HTML markup requires the client software to be able to parse or filter it. While this is fine with many desktop agents, it restricts developers looking for other uses of the data. This brings us to the philosophical aspect. RSS's second use, after providing headlines and content to desktop readers and sites, is to provide indexable metadata. By combining presentation and content (i.e., by including HTML markup within the description element), you could disable this feature.

However, my opinion lost out on this one. RSS 2.0 now allows for entity-encoded HTML within the `item/description` tag. It doesn't mention anything, in either direction, regarding `item/title`, and people are basically making it up as they go along. With that in mind, I still state that `item/title` at least should be considered plain text. If you want to put HTML within the `item/description` element, you can do it in two ways:

### *Entity encoding*

With entity encoding, the angle brackets of HTML tags are converted to their respective HTML entities, `&lt;` and `&gt;`. If you need to show angle brackets as literal characters, the ampersand character itself should be encoded as well:

This is a `&lt;em&gt;` lovely left angle bracket: `&lt;/em&gt;` &amp;lt;

### *Within a CDATA block*

The alternative is to enclose the HTML within a CDATA block. This removes one level of entity encoding, as in:

```
<![CDATA[This is a <em> lovely left angle bracket: </em> &lt;]]>
```

Either approach is acceptable according to the specification, and there is no way for a program to tell the difference between the two, or to tell if the description is actually just plain text that *resembles* encoded HTML. This is a major problem with the RSS 2.0 specification, as you'll see when we talk about parsing feeds. Atom and RSS 1.0 both have their own ways around this issue.

### language

The language the feed is written in. This allows aggregators to index feeds by language and should contain the standard Internet language codes as per RFC 1766.

```
<language>en-US</language>
```

### copyright

A copyright notice for the content in the feed:

```
<copyright>Copyright 2004 Ben Hammersley</copyright>
```

### managingEditor

The email address of the person to contact for editorial enquiries. It should be in the format: *name@example.com* (*FirstName LastName*).

```
<managingEditor>ben@benhammersley.com (Ben Hammersley)</managingEditor>
```

### webMaster

The email address of the person responsible for technical issues with the feed:

```
<webMaster>techsupport@benhammersley.com (Geek McNerdy)</webMaster>
```

### pubDate

The publication date of the content within the feed. For example, a daily morning newspaper publishes at a certain time early every morning. Technically, any information in the feed should not be displayed until after the publication date, so you can set `pubDate` to a time in the future and expect that the feed won't be displayed until after that time. Few existing RSS readers take any notice of this element in this way, however. Nevertheless, it should be in the format outlined in RFC 822:

```
<pubDate>Sun, 12 Sep 2004 19:00:40 GMT</pubDate>
```

### lastBuildDate

The date and time, RFC 822-style, when the feed last changed. Note the difference between this and `channel/pubDate`. `lastBuildDate` must be in the past. It is this element that feed applications should take as the "last time updated" value and not `channel/pubDate`.

```
<pubDate>Sun, 12 Sep 2004 19:01:55 GMT</pubDate>
```

### category

Identical in syntax to the `item/category` element you'll see later. This takes one optional attribute, `domain`. The value of `category` should be a forward-slash-separated string that identifies a hierarchical location in a taxonomy represented by the `domain` attribute. Sadly, there is no consensus either within the specification or in the real world as to any standard format for the `domain` attribute. It would seem most sensible to restrict it to a URL; however, it needn't necessarily be so.

```
<category domain="Syndic8">1765</category>
```

### generator

This should contain a string indicating which program created the RSS file:

```
<generator>Movable Type v3.1b3</generator>
```

### docs

A URL that points to an explanation of the standard for future reference. This should point to <http://blogs.law.harvard.edu/tech/rss>:

```
<docs>http://blogs.law.harvard.edu/tech/rss</docs>
```

## cloud

The `<cloud/>` element enables a rarely used feature known as “Publish and Subscribe,” which we shall investigate fully in Chapter 9. It takes no value itself, but it has five mandatory attributes, themselves also explained in Chapter 9: domain, path, port, registerProcedure, and protocol.

```
<cloud domain="rpc.sys.com" port="80" path="/RPC2" registerProcedure="pingMe"
protocol="soap"/>
```

## ttl

ttl, short for Time-to-Live, should contain a number, which is the minimum number of minutes the reader should wait before refreshing the feed from its source. Feed authors should adjust this figure to reflect the time between updates and the number of times they wish their feed to be requested, versus how up to date they need their consumers to be.

```
<ttl>60</ttl>
```

## image

This describes a feed’s accompanying image. It’s optional, but many aggregators look prettier if you include one. It has three required and two optional subelements of its own:

### url

The URL of a GIF, JPG, or PNG image that corresponds to the feed. It is, quite obviously, required.

### title

A description of the image, normally used within the ALT attribute of HTML’s `<img>` tag. It is required.

### link

The URL to which the image should be linked. This is usually the same as the channel/link.

### width and height

The width and height of the icon, in pixels. The icons should be a maximum of 144 pixels wide by 400 pixels high. The emergent standard is 88 pixels wide by 31 pixels high. Both elements are optional.

```
<image> <title>RSS2.0 Example</title> <url>http://www.exampleurl.com/example/
images/logo.gif</url> <link>http://www.exampleurl.com/example/index.html</link>
<width>88</width> <height>31</height> <description>The World’s Leading Technical
Publisher</description> </image>
```

## rating

The PICS rating for the feed; it helps parents and teachers control what children access on the Internet. More information on PICS can be found at <http://www.w3.org/PICS/>. This labeling scheme is little used at present, but an example of a PICS rating would be:

```
<rating>(PICS-1.1 "http://www.gcf.org/v2.5" labels on "1994.11.05T08:15-0500"
until "1995.12.31T23:59-0000" for "http://w3.org/PICS/Overview.html" ratings
(suds 0.5 density 0 color/hue 1))</rating>
```

## textInput

An element that lets RSS feeds display a small text box and Submit button, and associates them with a CGI application. Many RSS parsers support this feature, and many sites use it to offer archive searching or email newsletter sign-ups, for example. textInput has four required subelements:

### title

The label for the Submit button. It can have a maximum of 100 characters.

### description

Text to explain what the textInput actually does. It can have a maximum of 500 characters.

### name

The name of the text object that is passed to the CGI script. It can have a maximum of 20 characters.

### link

The URL of the CGI script.

```
<textInput> <title>Search</title> <description>Search the Archives</description> <name>query</name> <link>http://www.exampleurl.com/example/search.cgi</link> </textInput>
```

## skipDays and skipHours

A set of elements that can control when a feed user reads the feed. skipDays can contain up to seven day subelements: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, or Sunday. skipHours contains up to 24 hour subelements, the numbers 1–24, representing the time in Greenwich Mean Time (GMT). The client should not retrieve the feed during any day or hour listed within these two elements. The elements are ORed not ANDed: in the example here, the application is instructed not to request the feed during 8 p.m. on any day, and never on a Monday:

```
<skipDays><day>Monday</day></skipDays> <skipHours><hour>20</hour></skipHours>
```

## item Elements

RSS 2.0 can have any number of item elements. The item element is at the heart of RSS; it contains the primary content of the feed. Technically, item elements are optional, but a syndication feed with no items is just a glorified link. Not having any items doesn't mean the feed is invalid, just extremely boring.

All item subelements are optional, with the proviso that at least one of item/title or item/description is present. You can use this feature to build lists (more on that later).

With item, there are the 10 standard item subelements available:

## title

Usually, this is the title of the story linked to by the item, but it can also be seen as a one-line list item. There is controversy over whether HTML is allowed within this element; for more information, see the sidebar “Including HTML Within title or description.”

## link

The URL of the story the item is describing.

## description

A synopsis of the story. The description can contain entity-encoded HTML. Again, as with `item/title`, see the pertinent sidebar “Including HTML Within title or description.”

## author

This should contain the email address of the resource’s author referred to within the item. The specification’s example is in the format `user@example.com (firstname lastname)` but isn’t explained further:

```
<author>ben@benhammersley.com (Ben Hammersley)</author>
```

## category

Exactly the same as `channel/category`, but it pertains to the individual item only:

```
<category domain="the twisted passages of my mind">up/to_the_left/there</category>
```

## comments

This should contain the URL of any comments page for the item; it’s primarily used with weblogs:

```
http://www.example.com/comments.cgi?post=12345
```

## enclosure

This describes a file associated with an item. It has no content, but it takes three attributes: `url` is the URL of the enclosure, `length` is its size in bytes, and `type` is the standard MIME type for the enclosure. Some feed applications can download these files automatically. The original idea was for configuring a feed aggregator to automatically download large media files overnight, thereby deferring the extra bandwidth required. This is an underused feature of RSS 2.0 because most aggregators don’t support it, but in 2004, it became the focus of a lot of development around the idea of podcasting. See the sidebar “Including HTML Within title or description” for details.

```
<enclosure url="http://www.example.com/hotxxxpron.mpg" length= "34657834" type="video/mpeg"/>
```

## guid

Standing for Globally Unique Identifier, this element should contain a string that uniquely identifies the item. It must never change, and it must be unique to the object it is describing. If that content changes in any way, it must gain a new `guid`. This element also has the optional attribute `isPermalink`, which, if true, denotes that the value of the element can be taken as a URL to the object

referred to by the item. Therefore, if no `item/link` element is present, but the `isPermalink` attribute is set to `true`, the application can take the value of `guid` in its place. The specification doesn't say what to do if both are present and aren't the same, but it seems sensible to give preference within any application to the `item/link` element.

```
<guid isPermalink="true">http://www.example.com/example.html</guid>
```

#### pubDate

The publication date of the item. Again, as with `channel/pubDate`, any information in the `item` shouldn't be displayed until after the publication date, but few existing RSS readers take any notice of this element in this way. The date is in RFC 822 format.

```
<pubDate>Mon, 13 Sep 2004 00:23:05 GMT</pubDate>
```

#### source

This should contain the name of the feed of the site from which the `item` was derived, and the attribute `url` should be the URL of that other site's feed:

```
<source url="http://www.metafilter.com/rss.xml">Metafilter</source>
```

Example 4-1 shows these parts assembled into an RSS 2.0 XML document.

#### Example 4-1. An example RSS 2.0 feed

```
<?xml version="1.0"?>
<rss version="2.0">
<channel>
  <title>RSS2.0Example</title>
  <link>http://www.exampleurl.com/example/index.html</link>
  <description>This is an example RSS 2.0 feed</description>
  <language>en-gb</language>
  <copyright>Copyright 2002, O'Reilly and Associates.</copyright>
  <managingEditor>example@exampleurl.com</managingEditor>
  <webMaster>webmaster@exampleurl.com</webMaster>
  <rating> </rating>
  <pubDate>03 Apr 02 1500 GMT</pubDate>
  <lastBuildDate>03 Apr 02 1500 GMT</lastBuildDate>
  <docs>http://blogs.law.harvard.edu/tech/rss</docs>
  <skipDays><day>Monday</day></skipDays>
  <skipHours><hour>20</hour></skipHours>
  <category domain="http://www.dmoz.org">Business/Industries/Publishing/Publishers/
Nonfiction/Business/O'Reilly_and_Associates</category>
  <generator>NewsAggregator'o'Matic</generator>
  <ttl>30</ttl>
  <cloud domain="http://www.exampleurl.com" port="80" path="/RPC2"
registerProcedure="pleaseNotify" protocol="XML-RPC" />

  <image>
    <title>RSS2.0 Example</title>
    <url>http://www.exampleurl.com/example/images/logo.gif</url>
    <link>http://www.exampleurl.com/example/index.html</link>
    <width>88</width>
```

Example 4-1. An example RSS 2.0 feed (continued)

```
<height>31</height>
<description>The World's Leading Technical Publisher</description>
</image>

<textInput>
  <title>Search</title>
  <description>Search the Archives</description>
  <name>query</name>
  <link>http://www.exampleurl.com/example/search.cgi</link>
</textInput>

<item>
  <title>The First Item</title>
  <link>http://www.exampleurl.com/example/001.html</link>
  <description>This is the first item.</description>
  <source url="http://www.anothersite.com/index.xml">Another Site</source>
  <enclosure url="http://www.exampleurl.com/example/001.mp3" length="543210"
    type="audio/mpeg"/>
  <category domain="http://www.dmoz.org">Business/Industries/Publishing/Publishers/
Nonfiction/Business/0'Reilly_and_Associates/</category>
  <comments>http://www.exampleurl.com/comments/001.html</comments>
  <author>Ben Hammersley</author>
  <pubDate>Sat, 01 Jan 2002 0:00:01 GMT</pubDate>
  <guid isPermaLink="true">http://www.exampleurl.com/example/001.html</guid>
</item>

<item>
  <title>The Second Item</title>
  <link>http://www.exampleurl.com/example/002.html</link>
  <description>This is the second item.</description>
  <source url="http://www.anothersite.com/index.xml">Another Site</source>
  <enclosure url="http://www.exampleurl.com/example/002.mp3" length="543210"
    type="audio/mpeg"/>
  <category domain="http://www.dmoz.org">Business/Industries/Publishing/Publishers/
Nonfiction/Business/0'Reilly_and_Associates/</category>
  <comments>http://www.exampleurl.com/comments/002.html</comments>
  <author>Ben Hammersley</author>
  <pubDate>Sun, 02 Jan 2002 0:00:01 GMT</pubDate>
  <guid isPermaLink="true">http://www.exampleurl.com/example/002.html</guid>
</item>

<item>
  <title>The Third Item</title>
  <link>http://www.exampleurl.com/example/003.html</link>
  <description>This is the third item.</description>
  <source url="http://www.anothersite.com/index.xml">Another Site</source>
  <enclosure url="http://www.exampleurl.com/example/003.mp3" length="543210"
    type="audio/mpeg"/>
  <category domain="http://www.dmoz.org">Business/Industries/Publishing/Publishers/
Nonfiction/Business/0'Reilly_and_Associates/</category>
  <comments>http://www.exampleurl.com/comments/003.html</comments>
  <author>Ben Hammersley</author>
```

*Example 4-1. An example RSS 2.0 feed (continued)*

```
<pubDate>Mon, 03 Jan 2002 0:00:01 GMT</pubDate>
<guid isPermaLink="true">http://www.exampleurl.com/example/003.html</guid>
</item>
</channel>
</rss>
```

## The Simplest Possible RSS 2.0 Feed

This, really, is the key to the success of RSS 2.0. The simplest thing you need to do to make the feed validate is very uncomplicated indeed (see Example 4-2). While this isn't any help when you are trying to convey complex information, as with RSS 1.0, or if you're trying to build a complete document-centric system, as with Atom, it is very useful for many other applications.

*Example 4-2. The simplest possible RSS 2.0 feed*

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0">

<channel>
<title>The Simplest Feed</title>
<link>http://example.org/index.html</link>
<description>The Simplest Possible RSS 2.0 Feed</description>

<item>
<description>Simple Simple Simple</description>
</item>
</channel>
</rss>
```

Chapter 10 describes many useful applications that take this a minimalist approach to using RSS 2.0—compliant feeds.

## Producing RSS 2.0 with Blogging Tools

The vast majority of RSS 2.0 feeds are produced by weblogging tools that use templates. The most popular of these is Movable Type, written by Ben and Mena Trott, which is freely available for personal use at <http://www.movabletype.org>. In order to discuss a few important implementation points, Example 4-3 shows a template for Movable Type that produces an RSS 2.0 feed.

*Example 4-3. A Movable Type template for producing RSS 2.0*

```
<?xml version="1.0"?>
<rss version="2.0">
<channel>
<title><$MTBlogName$></title>
<link><$MTBlogURL$></link>
```

Example 4-3. A Movable Type template for producing RSS 2.0 (continued)

```
<description><$MTBlogDescription$></description>
<language>en-gb</language>
<copyright>All content Public Domain</copyright>
<managingEditor>ben@benhammersley.com</managingEditor>
<webMaster>ben@benhammersley.com</webMaster>
<docs>http://blogs.law.harvard.edu/tech/rss</docs>
<category domain="http://www.dmoz.org">Reference/Libraries/Library_and_Information_
Science/Technical_Services/Cataloguing/Metadata/RDF/Applications/RSS/</category>
<generator>Movable_Type/2.5</generator>
<lastBuildDate><$MTDate format="%a, %d %b %Y %I:%M:00 GMT"$></lastBuildDate>
<ttl>60</ttl>

<MTEntries lastn="15">
<item>
<title><$MTEntTitle encode_html="1"$></title>
<description><$MTEntExcerpt encode_html="1"$></description>
<link><$MTEntLink$></link>
<comments><$MTEntLink$></comments>
<author><$MTEntAuthorEmail$></author>
<pubDate><$MTEntDate format="%a, %d %b %Y %I:%M:00 GMT"$></pubDate>
<guid isPermaLink="false">GUID:<$MTEntLink$></g><$MTEntDate format=
"%a%d%b%Y%M"$></guid>
</item>
</MTEntries>
</channel>
</rss>
```

The vast majority of this template is standard Movable Type fare. Taken from one of my own blogs, it uses the `<$MT$>` tags to insert information directly from the Movable Type database into the feed. So far, so simple.

Two things are worth close examination. First, the date format:

```
<pubDate><$MTEntDate format="%a, %d %b %Y %I:%M:00 GMT"$></pubDate>
```

Care must be taken to ensure that the format of the contents of the date fields are correctly formed. RSS 2.0 feeds require their dates to be written to comply with RFC 822—for example: Mon, 03 Jan 2002 0:00:01 GMT.

Common errors found in RSS 2.0 feeds include missing commas, seconds values, and time zones. You must ensure these are all present because some desktop readers and aggregators aren't as forgiving as others, and many are getting less so as they develop.

Implementation of the `guid` element is equally important. The RSS 2.0 standard doesn't discuss the form of the `guid`; it only asks the author to ensure that it is globally unique. There is no scope for the OSF GUID standard to be used within most blogging tools, so you have to formulate your own system. (I'll touch upon this again when we talk about Atom in Chapter 7.)

For the template shown earlier, I considered various things. First, the `guid`'s purpose is to tell applications if the entry is new or if it has changed. Second, within my own blogs I allow people to add comments to the entries. I consider this a change to the entry, so my `guid` must reflect this. Because this change isn't reflected in the link to the entry, the link alone isn't a good `guid`. So, by combining the link with the last-updated-date value, I can make a `guid` that is globally unique and changes when it needs to. For added measure, I add the string `GUID` to the front of it to prevent it from looking too much like a retrievable URL—which, of course, it isn't. Hence:

```
<guid isPermaLink="false">GUID:<$MEntryLink$></g<$MEntryDate format=
"%a%d%b%Y%I:%M"$></guid>
```

This works well as an RSS 2.0 `guid`, but it has one feature that might annoy: because weblog comments cause the `guid` to change, it also causes the item to be marked as unread in many feed aggregators. This might not be desirable behavior for you.

## Introducing Modules

Modules are additional sets of elements, giving the feed a greater range of expression: they allow the specification to be extended without actually being changed, which is a very clever trick. You can make your own module match any data you might wish to syndicate. Admittedly, most aggregators will ignore it, but your own applications can take advantage of it. And, happily, the most popular modules are increasingly being supported by the latest aggregators as a matter of course.

Modules in RSS, both Versions 2.0 and 1.0, are created with a system known as XML Namespaces. Namespaces are the XML solution to the classic language problem of one word meaning two things in different contexts. Take “Windows,” for example. In the context of houses, “windows” are holes in the wall through which we can look. In the context of computers, “Windows” is a trademark of the Microsoft Corporation and refers to its range of operating systems. The context within which the name has a particular meaning is called its namespace.

In XML, you can distinguish between the two meanings by assigning a namespace and placing the namespace's name in front of the element name, separated by a colon, like this:

```
<computing:windows>This is an operating system</computing:windows>

<building:windows>This is a hole in a wall</building:windows>
```

Namespaces solve two problems. First, they allow you to distinguish between different meanings for words that are spelled the same way, which means you can use words more than once for different meanings. Second, they allow you to group together words that are related to each other; for example, using a computer to look through an XML document for all elements with a certain namespace is easy.

Both RSS 1.0 and 2.0 use namespaces to allow for *modularization*. This modularization means that developers can add new features to RSS documents without changing the core specification.

Modularization has great advantages over the older RSS 0.9x's method for including new elements. For starters, anyone can create a module: there are no standards issues or any need for approval, aside from making sure that the namespace URI you use has not been used before. And, it means both RSS 1.0 and 2.0 are potentially far more powerful than RSS 0.9x ever was.

A module works in the actual RSS document by declaring a namespace within the root element of the feed and by prefixing the element's names with that namespace prefix, like so:

```
<?xml version="1.0"?>
<rss version="2.0" xmlns:blogChannel="http://backend.userland.com/blogChannelModule">
...

  <blogChannel:blink>http://www.benhammersley.com</blogChannel:blink>
...

```

You should note that the URI the namespace declaration points to is the unique identifier of the namespace and not the namespace prefix. In other words, from the perspective of a program processing XML, this:

```
<?xml version="1.0"?>
<rss version="2.0" xmlns:blogChannel="http://backend.userland.com/blogChannelModule">
...

  <blogChannel:blink>http://www.benhammersley.com</blogChannel:blink>
...

```

is absolutely identical to this:

```
<?xml version="1.0"?>
<rss version="2.0" xmlns:bingbangbong="http://backend.userland.com/
blogChannelModule">
...

  <bingbandbong:blink>http://www.benhammersley.com</bingbangbong:blink>...

```

This will become clear as we study some common modules. It is customary, and also very good manners, to have documentation for the module to be found at the namespace's URI, but this isn't technically necessary. As discussed in Chapter 11, the different feed standards have different scopes for the form this documentation can take. The presence of anything at all at the namespace URI is entirely optional, both in terms of RSS and within the scope of the broader XML specification itself.

## blogChannel Module

Designed by Dave Winer only a week after he formalized RSS 2.0, the `blogChannel` module allows the inclusion of data used by weblogging applications and, specifically, the newer generation of aggregating and filtering systems.

It consists of three optional elements, all of which are subelements of `channel` and have the following namespace declaration:

```
xmlns:blogChannel="http://backend.userland.com/blogChannelModule"
```

The elements are:

`blogChannel:blogRoll`

Contains a literal string that is the URL of an OPML file containing the `blogRoll` for the site. A *blogroll* is the list of blogs the blog author habitually reads.

`blogChannel:blink`

Contains a literal string that is the URL of a site the blog author recommends the reader visits.

`blogChannel:mySubscriptions`

Contains a literal string that is the URL of the OPML file containing the URLs of the RSS feeds to which the blog author is subscribed in her desktop reader.

Example 4-4 shows the beginning of an RSS 2.0 feed using the `blogChannel` module.

*Example 4-4. An RSS 2.0 feed with the `blogChannel` module*

```
<?xml version="1.0"?>
<rss version="2.0" xmlns:blogChannel="http://backend.userland.com/blogChannelModule">
<channel>
  <title>RSS2.0Example</title>
  <link>http://www.exampleurl.com/example/index.html</link>
  <description>This is an example RSS 2.0 feed</description>
  <blogChannel:blogRoll>http://www.exampleurl.com/blogroll.opml</blogChannel:blogRoll>
  <blogChannel:blink>http://www.benhammersley.com</blogChannel:blink>
  <blogChannel:mySubscriptions>http://www.exampleurl.com/mySubscriptions.opml
</blogChannel:mySubscriptions>
...

```

## Creative Commons Module

Also designed by Dave Winer, the Creative Commons module allows RSS 2.0 feeds to specify which Creative Commons license applies to them. The Creative Commons organization, <http://creativecommons.org/>, offers a variety of content licenses that allow feed publishers to release content under more flexible copyright restrictions than previously available. Feed consumers can consult the license to see how they can reuse the content for their own work.

The element can apply to either the complete channel or the individual item.

It consists of only one element, `creativecommons:license`, which contains the URL of the Creative Commons license on the Creative Commons site. It has the following namespace declaration:

```
xmlns:creativecommons="
http://backend.userland.com/creativecommonsRssModule"
```

In action, it looks like Example 4-5.

*Example 4-5. Part of an RSS 2.0 feed with the Creative Commons module*

```
<rss version="2.0" xmlns:creativecommons="http://backend.userland.com/
creativecommonsRssModule">

<channel>
<title>Creative Commons Example</title>
<link>http://www.example.com/</link>
<creativecommons:license>http://www.creativecommons.org/licenses/by-nd/1.0
</creativecommons:license>
...
<item>
<description>blah blah blah</description>
<creativecommons:license>http://www.creativecommons.org/licenses/by-nc/1.0
</creativecommons:license>
</item>
...
```

Note that a `creativecommons:license` element on an `item` overrides the same on the channel for that item.

More details can be found at:

<http://backend.userland.com/creativecommonsRssModule>

## Simple Semantic Resolution Module

One of the never-ending arguments within the RSS world is that between the pro- and anti-RDF camps. The fork between RSS 0.91 and 1.0 was almost entirely caused by this disagreement. The pro-RDF camp stated, quite rightly, that RDF data has a great deal more meaningful utility than plain XML, whilst the anti-RDF camp stated, also quite rightly, that the RDF syntax was horrible, and that no one can understand it without reading the documentation and having a nice lie down.

That may be—we’ll find out your own feelings on this in the next chapters—but in the meantime, the Simple Semantic Resolution module was one idea put forward to bridge the divide between the two cultures.

Written by Danny Ayers, its presence in an RSS 2.0 feed simply means “this data should be considered RDF, and to use it with an RDF-compatible application you should apply this transformation to it first.” Whereupon, it points you to a nice

XSLT stylesheet. That stylesheet consists of one single element, a subelement of channel, and has the following namespace declaration:

```
xmlns:ssr="http://purl.org/stuff/ssr"
```

The element is:

```
ssr:rdf
```

It's empty, but contains a single attribute, transform, which is equal to the URL of the necessary stylesheet:

```
<ssr:rdf transform="http://w3future.com/weblog/gems/rss2rdf.xml" />
```

Example 4-6 shows the SSR module in use.

*Example 4-6. Part of an RSS 2.0 feed with the SSR module*

```
<?xml version="1.0"?>
<rss version="2.0" xmlns:ssr="http://purl.org/stuff/ssr">
<ssr:rdf transform="http://w3future.com/weblog/gems/rss2rdf.xml" />
...

```

More details can be found at <http://ideagraph.net/xmlns/ssr/>.

## Trackback Module

The trackback system for weblog content management systems (see <http://www.movabletype.org/docs/mtrackback.html> for the technical details) has grown up in the same neighborhood as RSS, so it's only fair that the one should be represented in the other.

This module, also available in tasty RSS 1.0, comes from Justin Klubnik and allows RSS 2.0 feeds to display both the URL that people should trackback to, but also the URL that the item has trackbacked itself. The idea is that aggregators can send pings and also follow links to find related pages, because items might ping places they don't explicitly link to.

This module is made up of two elements, subelements of item, and has the following namespace declaration:

```
xmlns:trackback="http://madskills.com/public/xml/rss/module/trackback/"
```

Here are the elements:

```
trackback:ping
```

This contains the item's trackback URL:

```
<trackback:ping>http://foo.com/trackback/tb.cgi?tb_id=20020923</trackback:ping>
```

```
trackback:about
```

This contains any trackback URL that was pinged in reference to the item:

```
<trackback:about>http://foo.com/trackback/tb.cgi?tb_id=20020923</trackback:about>
```

More details can be found at <http://madskills.com/public/xml/rss/module/trackback/>.

## ICBM Module

This module, written by Matt Croydon and Kenneth Hunt, allows RSS feeds to state the geographical location of the origin of the feed or an individual item within it.

It's alleged that ICBM does actually stand for intercontinental ballistic missile, and certainly a half-arsed attempt at Googling for it produces only the explanation that describing one's position as an ICBM address is so that, should anyone wish, your data will allow the baddies to target you directly, presumably for being far too clever with your syndication feeds.

Either way, the namespace declaration is thus:

```
xmlns:icbm="http://postneo.com/icbm"
```

It contains two elements, usable in either the channel or the item context. The item context overrides the former, as you might expect.

`icbm:latitude`

This contains the latitude value as per the geographic standard WGS84:

```
<icbm:latitude>43.7628</icbm:latitude>
```

`icbm:longitude`

This contains the longitude value as per the geographic standard WGS84.

```
<icbm:longitude>11.2442</icbm:longitude>
```

That's my house, actually.

Go to <http://www.postneo.com/icbm/> for more verbose details on the thinking behind the specification.

## Yahoo!'s Media RSS Module

In December 2004, Yahoo! launched a beta video search engine at <http://video.search.yahoo.com/>. The original system spidered the Web looking for video files and indexed them with the implied information found in the filename and link text. To make it easier for video content producers to have Yahoo! index their sites, and to give the search engine much better data to play with, Yahoo! is now offering to regularly spider RSS feeds containing details of media files. This additional data is encoded in its new Media RSS Module.

That module consists of one element, `<media:content>`, with a namespace declaration of:

```
xmlns:media="http://tools.search.yahoo.com/mrss/"
```

and four optional subelements. `<media:content>` is a subelement of `item` and consists of ten optional attributes.

`url`

`url` specifies the direct URL to the media object. It is an optional attribute. If a URL isn't included, a `playerURL` must be specified.

`fileSize`

The size, in bytes, of the media object. It is an optional attribute.

`type`

The standard MIME type of the object. It is an optional attribute.

`playerURL`

`playerURL` is the URL of the media player console. It is an optional attribute.

`playerHeight`

`playerHeight` is the height of the window the `playerURL` should be opened in. It is an optional attribute.

`playerWidth`

`playerWidth` is the width of the window the `playerURL` should be opened in. It is an optional attribute.

`isDefault`

`isDefault` determines if this is the default object that should be used for this element. It can be `true` or `false`. So, if an `item` contains more than one `media:content` element, setting this to `true` makes it the default. It's an optional attribute but can be used only once within each `item`.

`expression`

`expression` determines if the object is a sample or the full version of the object. It can be either `sample` or `full`. It is an optional attribute.

`bitrate`

The bit rate of the file, in kilobits per second. It is an optional attribute.

`duration`

The number of seconds the media plays, for audio and video. It is an optional attribute.

There are also four optional subelements to `<media:content>`, which can be also used as subelements to `item`:

`<media:thumbnail>`

Allows a particular image to be used as the representative image for the media object:

```
<media:thumbnail height="50" width="50">  
  http://www.foo.com/keyframe.jpg</media:thumbnail>
```

It takes two optional attributes. `height` specifies the height of the thumbnail. `width` specifies the width of the thumbnail.

<media:category>

Allows a taxonomy to be set that gives an indication of the type of media content and its particular contents:

```
<media:category>music/artist name/album/song</media:category>
<media:category>television/series/episode/episode number</media:category>
```

<media:people>

Lists the notable individuals or businesses and their contribution to the creation of the media object.

```
<media:people role="editor">Simon St Laurent</media:people>
```

role specifies the role individuals played. Examples include: producer, artist, news anchor, cast member, etc. It is an optional attribute.

<media:text>

Allows the inclusion of a text transcript, closed captioning, or lyrics of the media content:

```
<media:text>Oh, say, can you see, by the dawn's early light,</media:text>
```

Once your site has a feed working with the Media RSS Module, like that shown in Example 4-7, you can submit it to Yahoo! at <http://tools.search.yahoo.com/mrss/submit.html>.

*Example 4-7. media:content in action*

```
<media:content url="http://www.example.com/movie.mov" fileSize="12345678" type=
  "video/quicktime"
  playerUrl="http://http://www.example.com/player?id=1" playerHeight="200"
  playerWidth="400"
  isDefault="true" expression="full" bitrate="128" duration="185">
  <media:thumbnail height="50" width="50">http://www.example.com/thumbnail.jpg
  thumbnail</media:

  <media:category>comedy/slapstick/custard</media:category>
  <media:people role="stuntman">Ben Hammersley</media:people>
  <media:text>Take that! And that! And that!</media:text>
</media:content>
```

The development of your own modules is covered in Chapter 11.

## Creating RSS 2.0 Feeds

RSS 0.91 and 0.92 feeds are created in the same way; the additional elements found in 0.92 are well-handled by the existing RSS tools.

Of course, you can always hand-code your RSS feed. Doing so certainly gets you on top of the standard, but it's neither convenient, quick, nor recommended. Ordinarily, feeds are created by a small program in one of the scripting languages: Perl, PHP, Python, etc. Many CMSs already create RSS feeds automatically, but you may want to create a feed in another context. Hey, you might even write your own CMS!

There are various ways to create a feed, all of which are used in real life:

### *XML transformation*

Running a transformation on an XML master document converts the relevant parts into RSS. This technique is used in Apache Axkit-based systems, for example.

### *Templates*

You can substitute values within a RSS feed template. This technique is used within most weblogging platforms, for example.

### *An RSS-specific module or class within a scripting language*

This method is used within hundreds of little ad hoc scripts across the Net, for example.

We'll look at all three methods, but let's start with the third, using an RSS-specific module. In this case, it's Perl's `XML::RSS`.

## **Creating RSS with Perl Using XML::RSS**

The `XML::RSS` module is one of the key tools in the Perl RSS world. It is built on top of `XML::Parser`—the basis for many Perl XML modules—and is object-oriented. Actually, `XML::RSS` also supports the creation of the older versions of RSS, plus RSS 1.0, and it can parse existing feeds, but in this section we will deal only with its 2.0 creation capabilities.

Incidentally, `XML::RSS` is an open source project. You can lend a hand, and grab the latest version, at <http://sourceforge.net/projects/perl-rss>.

Examples 4-8 and 4-9 show a simple Perl script and the feed it creates.

### *Example 4-8. A sample XML::RSS script*

```
#!/usr/bin/perl

use Warnings;
use Strict;
use XML::RSS;

my $rss = new XML::RSS( version => '2.0' );

$rss->channel(
    title      => 'The Title of the Feed',
    link       => 'http://www.oreilly.com/example/',
    language   => 'en',
    description => 'An example feed created by XML::RSS',
    lastBuildDate => 'Tue, 14 Sep 2004 14:30:58 GMT',
    docs       => 'http://blogs.law.harvard.edu/tech/rss',
);

$rss->image(
    title      => 'Oreilly',
    url        => 'http://meerkat.oreillynet.com/icons/meerkat-powered.jpg',
```

*Example 4-8. A sample XML::RSS script (continued)*

```
    link      => 'http://www.oreilly.com/example/',
    width     => 88,
    height    => 31,
    description => 'A nice logo for the feed'
);

$rss->textinput(
    title     => "Search",
    description => "Search the site",
    name      => "query",
    link      => "http://www.oreilly.com/example/search.cgi"
);

$rss->add_item(
    title     => "Example Entry 1",
    link      => "http://www.oreilly.com/example/entry1",
    description => 'blah blah',
);

$rss->add_item(
    title     => "Example Entry 2",
    link      => "http://www.oreilly.com/example/entry2",
    description => 'blah blah'
);

$rss->add_item(
    title     => "Example Entry 3",
    link      => "http://www.oreilly.com/example/entry3",
    description => 'blah blah'
);

print $rss->as_string;
```

*Example 4-9. The resultant RSS 2.0 feed*

```
<?xml version="1.0" encoding="UTF-8"?>

<rss version="2.0" xmlns:blogChannel="http://backend.userland.com/blogChannelModule">

<channel>
<title>The Title of the Feed</title>
<link>http://www.oreilly.com/example/</link>
<description>An example feed created by XML::RSS</description>
<language>en</language>
<lastBuildDate>Tue, 14 Sep 2004 14:30:58 GMT</lastBuildDate>
<docs>http://blogs.law.harvard.edu/tech/rss</docs>

<image>
<title>Oreilly</title>
<url>http://meerkat.oreillynet.com/icons/meerkat-powered.jpg</url>
<link>http://www.oreilly.com/example/</link>
<width>88</width>
```

*Example 4-9. The resultant RSS 2.0 feed (continued)*

```
<height>31</height>
<description>A nice logo for the feed</description>
</image>

<item>
<title>Example Entry 1</title>
<link>http://www.oreilly.com/example/entry1</link>
<description>blah blah</description>
</item>

<item>
<title>Example Entry 2</title>
<link>http://www.oreilly.com/example/entry2</link>
<description>blah blah</description>
</item>

<item>
<title>Example Entry 3</title>
<link>http://www.oreilly.com/example/entry3</link>
<description>blah blah</description>
</item>

<textInput>
<title>Search</title>
<description>Search the site</description>
<name>query</name>
<link>http://www.oreilly.com/example/search.cgi</link>
</textInput>
</channel>
</rss>
```

After the required Perl module declaration, you create a new instance of `XML::RSS`, like so:

```
my $rss = new XML::RSS (version => '2.0');
```

The new method function returns a reference to the new `XML::RSS` object. The function can take three arguments, two of which are of interest here:

```
new XML::RSS (version=>$version, encoding=>$encoding);
```

The `version` attribute refers to the version of RSS you want to make (either '2.0' or '1.0', or, if you fancy being a bit retro, '0.91'), and the `encoding` attribute sets the encoding of the XML declaration. The default encoding, as with XML, is UTF-8.

The rest of the script is quite self-explanatory. The methods `channel`, `image`, `textInput`, and `add_item` all add new elements and associated values to the feed you are creating, and the `print $rss->as_string;` prints out the result. You can also call the `$rss->save` method to save the created feed as a file.

## guid, Permalink or not

XML::RSS does support the two `guid` `isPermalink` options but in a slightly less predictable way than the other element functions. To set `guid isPermalink="true"`, you should do this:

```
$rss->add_item(  
    title    => "Example Entry 1",  
    link     => "http://www.oreilly.com/example/entry1",  
    description => 'blah blah',  
    permalink => "http://www.oreilly.com/example/entry1",  
);
```

However, to set `guid isPermalink="false"`, you should do this:

```
$rss->add_item(  
    title    => "Example Entry 1",  
    link     => "http://www.oreilly.com/example/entry1",  
    description => 'blah blah',  
    guid     => "http://www.example.com/guid Rus/348324327",  
);
```

## Module support under XML::RSS

As you can see, XML::RSS always includes the namespace declaration for the `blogChannel` module. You can also use it to include other modules within your feed.

In Example 4-4, we passed known strings to the module. It's really not of much use as a script; you need to add a more dynamic form of data, or the feed will be very boring indeed. We do an awful lot of this sort of thing in Chapter 10, so let's leave Perl until then, and move on to another language.

## Creating RSS 2.0 with PHP

Great RSS 2.0 support for PHP is to be found in the `feedcreator.class` by Kai Blankenhorn at <http://www.bitfolge.de>. Unlike the previous section's XML::RSS, `feedcreator.class` can only create RSS feeds; it can't parse them. No matter: it's very good at that indeed.

As illustrated in Example 4-10, the function for each feed element is named after the element, so it behaves pretty much as you would expect.

*Example 4-10. A PHP script using FeedCreator that produces RSS 2.0*

```
<?php  
include("feedcreator.class.php");  
  
$rss = new UniversalFeedCreator();  
$rss->title = "Example Feed";  
$rss->description = "This is the feed description";  
$rss->link = "http://www.example.com/";
```

*Example 4-10. A PHP script using FeedCreator that produces RSS 2.0 (continued)*

```
// Image section
$image = new FeedImage();
$image->title = "example logo";
$image->url = "http://www.example.com/images/logo.gif";
$image->link = "http://www.example.com";
$image->description = "Visit Example.com!";
$rss->image = $image;

// Item Loop
$item = new FeedItem();
$item->title = "Entry one";
$item->link = "http://www.example.com/entryone";
$item->description = "This is the content of the first entry";
$item->author = "Ben Hammersley";
$rss->addItem($item);
// End Item Loop

?>
```

This is a very simple script. As you can see from the resulting feed, Example 4-11, it produces only one item. We'll be using it for more complicated things later on in the book, so in the meantime, once we've passed over the example output, we'll take a quick look at some of the special features.

*Example 4-11. An RSS 2.0 feed created with PHP*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- generator="FeedCreator 1.7.1" -->
<rss version="2.0">
  <channel>
    <title>Example Feed</title>
    <description>This is the feed description</description>
    <link>http://www.example.com/</link>
    <lastBuildDate>Thu, 16 Sep 2004 20:16:22 +0100</lastBuildDate>
    <generator>FeedCreator 1.7.1</generator>
    <image>
      <url>http://www.example.com/images/logo.gif</url>
      <title>example logo</title>
      <link>http://www.example.com/</link>
      <description>Visit Example.com!</description>
    </image>
    <item>
      <title>Entry one</title>
      <link>http://www.example.com/entryone</link>
      <description>This is the content of the first entry</description>
      <author>Ben Hammersley</author>
    </item>
  </channel>
</rss>
```

## Caching and saving

One advantage that the FeedCreator class has over Perl's XML::RSS is the built-in caching mechanism. PHP is mostly used, in this case, as a way of dynamically building a feed upon request, perhaps from a database. However, when a feed gets too popular, that might cause too much of a server load. You can have your script store a cache file and serve that instead of running itself by adding this line:

```
$rss->useCached();
```

This saves the dynamically created feed to a cache, serving that instead if it is less than one hour old. Remember, you need to place this line right underneath the `$rss = new UniversalFeedCreator();`, or you'll waste precious processor cycles.

You can also explicitly save the file with a command like this:

```
echo $rss->saveFeed("RSS2.0", "index.xml");
```

## Dates

Because, caching notwithstanding, the feeds are usually produced dynamically, FeedCreator declares the `channel/lastBuildDate` element automatically at the time of creation. You can, of course, specify it explicitly, as you can with `pubDate`. FeedCreator allows the use of RFC 822 (Mon, 20 Jan 03 18:05:41 +0400), ISO 8601 (2003-01-20T18:05:41+04:00), and Unix (1043082341) time values.

## Namespaced modules

This is the major drawback with the class. You can't, as of Version 1.71 at least, create a feed with modules in it. If you're set on doing that—perhaps with some groovy special in-house application in mind—you'll need to hack at the class's code. It is licensed under the GPL, so go right ahead.

## Creating RSS 2.0 with Ruby

Since Version 1.8.2, Ruby has shipped with Kouhei Sutou's RSS parsing and creation library. At time of writing, however, Ruby has only reached 1.8.2.preview.3, and documentation is hard to come by. The only documentation for the new RSS classes is found at:

*<http://www.cozmixng.org/~rwiki/?cmd=view;name=RSS+Parser%3A%3ATutorial.en>*  
in a potentially unreliable translation from the Japanese original.

Having said that, the library does seem very complete indeed, with support for the parsing and writing of both RSS 1.0 and 2.0. At time of writing, the tutorial just mentioned was growing rapidly and being completed by the library's author. Ruby programmers should check the URL for changes.

## Serving RSS 2.0

Although, or perhaps because, there is no official word within the specification regarding this, the growing standard for serving RSS 2.0 is with a MIME type of `application/xml`. Dave Winer prefers `text/xml` for the way that it causes the file to display itself nicely inside Internet Explorer. Using `application/xml` is more correct, but it causes browsers to download the file instead of displaying it. Really advanced users are looking at `application/rss+xml`, but currently no standard exists. It's up to you, but certainly, it should not be served with any other MIME type. `text/plain` is right out.