

# Writing an RSS Aggregator Part 1

## Building an application



By Jeffrey Houser

**S**o often in this column I feel that I'm writing about basic concepts and using trivial examples.

It's often up to you, as the reader, to figure out how to apply these concepts to your development. I thought it might be a good

idea to take some space to try to bring a lot of the concepts together and build an application. I don't always take the space to do so, but thought I'd give it a try here.

I'm sure that most of you read (or at least have heard of) blogs. Blogs are a great way to keep up on the up-to-the-minute comings and going of whatever community you choose. There are tons of bloggers throughout the ColdFusion community, some blogging from inside Adobe. Many blogs provide an RSS feed for content. RSS stands for Really Simple Syndication and is an XML dialect. There are a handful of RSS aggregators in the ColdFusion community such as mxna (<http://weblogs.macromedia.com/mxna/>), Full As a Goog (<http://www.fullasagoog.com/>), and Feed Squirrel (<http://www.feed-squirrel.com/>). I've always thought it would be cool to create something like a LiveJournal friends list of my very own. In this series of articles, I'm going to step you through the thought process that I'd go through to create one. First I'll concentrate on designing the database and fleshing out the ColdFusion components.

### Designing the Back End

When developing an application, I normally start with the database design, and that is where we'll start here. If you need a refresher on database design techniques, you can review the database design article from this column at <http://coldfusion.sys-con.com/read/49177.htm>. What data do you want to store in an RSS Aggregator application? You can read all about the RSS 2.0 spec at <http://www.rssboard.org/rss-specification>. For purposes of this article, I'm only going to concentrate on required fields.

- *RSS Feeds*: The RSS feed seems like a good place to start. The app will have to store the name of the RSS feed and the URL

location of the feed.

- *Items*: Since we're aggregating feed data from a lot of different sources, we'll need to store the posts from each individual feed. It'll be more efficient to store them locally than to create the feed manually each time we want to display something. A post is put into the RSS XML document as an item. According to the spec, nothing is required in the item except for title or description; there's no guarantee that one will be there. For the sake of this article, we'll assume that title, link, and description will be there. Items have a one-to-many relationship with RSS feeds. Each blog can contain many posts, but each post belongs to only one blog. To implement this in the database, we put a primary key for the RSS feed into the items table.
- *Categories*: Within the app, I want to be able to categorize each feed, so I can filter the group of messages that I'm reading. We'll add a category table to the database. This data won't be coming from the RSS, but some identifier that we create.

The database structure is shown in Figure 1.

For the model portion of this app, I'm going to use ColdFusion components. You can read more about component basics in two of my articles at <http://coldfusion.sys-con.com/read/47203.htm> and <http://coldfusion.sys-con.com/read/47446.htm>. I'd create one CFC for each element: an RSS Feed, an Item, and a Category. You can see the model in Figure 2. You may notice that the CFCs are named in the singular tense, while the database tables are plural. This is because each CFC is designed to represent a single item, while the database tables contain many items. I left out individual getter and setter methods from the diagram, but the code will use the generic getter and setter methods from Hal Helm's BaseComponent.cfc <http://halhelms.com/webresources/BaseComponent.cfc>. Each component includes an init and a commit method. The init will load information out of the database, while the commit will create (or update) information in the database.



Figure 1 Database structure

The RSSFeed and RSSCategory components will be used for maintaining feeds and categories. You might notice that the RSSFeed component contains an RSSCategory component instead of the CategoryID foreign key. I thought about putting an array of item objects in the RSSFeed, but decided against it. The application won't need to load all the items from the feed in question; a gateway object can be used to handle this type of action when viewing the data (although that's beyond the scope of this article).

The RSSAggregator component is where most of the magic takes place. It will be the background process, run as part of a scheduled task. It will loop through each feed in the database, collect the latest data, and store the new items in the database. It will use the item component

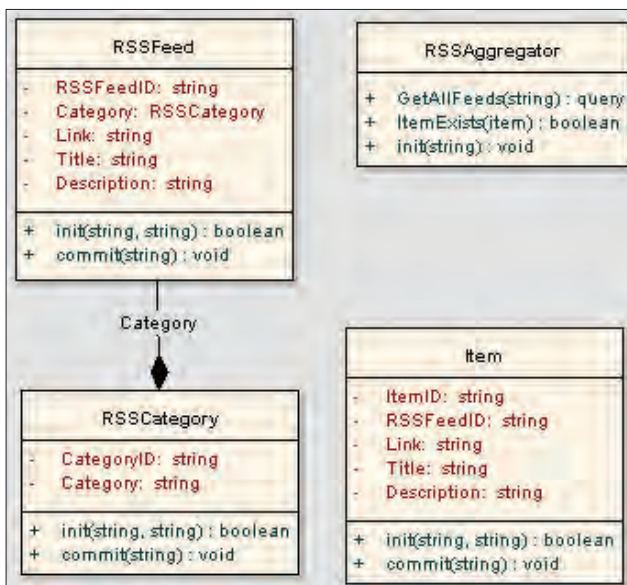


Figure 2: Object model

## Creating the RSS Feed Components

In real-world applications, I often like to start by building the administrator first. That way I don't have to worry about filling the tables with fake data to test the public-facing part of the site. Before we can start to process feeds on the back-end, we have to be able to enter the feeds into the database. We can start by fleshing out the RSSFeed and RSSCategory components.

First, let's examine the RSSCategory component, as shown in Listing 1. Nothing in the RSSCategory.cfc should surprise you. It contains two instance variables, the CategoryID and Category. The CategoryID is initialized as a string because this app uses UUIDs as primary keys instead of auto-incrementing integers. The component has two methods (plus a few inherited from Hal's BaseComponent.cfc). There's an init method that accepts the CategoryID and the datasource. It queries the database and sets the instance variables, if applicable. The second method is a commit method. It accepts the datasource name as an argument. If the CategoryID is blank, a new category is created. If it isn't, an update is performed. This isn't much different, in concept, from the address.cfc I wrote about in one of my previous

component articles.

The RSSFeed component is shown in Listing 2. You'll notice many similarities to the RSSCategory component. There are more properties in RSSFeed than RSSCategory, but the concept is the same. You can notice that the category property of the component is an instance of the RSSCategory component. Instead of containing the category information directly, only the reference is stored. To access the category information, we can use the get, or set, methods off the component instance. The set methods are demonstrated in init; while the get methods are demonstrated in commit.


## Entering the Data

To enter the data, a simple HTML form is used, as shown in Listing 3. The first lines include a syslib.cfm file. This is just a utility library of functions. Two functions from the library are used in this component. The GetCategories function contains a query to get all categories. The "CreateSelectList" function is very similar to the cselect tag for those who don't use cform. (I'll make sure that all the code behind this article gets up on my blog; I left out the syslib.cfm for brevity).

The file creates an instance of the RSSFeed component. If an RSSFeedID is defined, it will run the init method; otherwise the base component is used. A form is created that asks for the category, and the http link of the RSS feed. The title and description are displayed on this form, but are actually populated from the RSS feed, so they're not editable.

The form submits onto Feedip.cfm, which is shown in Listing 4. The initialization code is the same as in Listing 3. Then the code enters a try block. To get the description and title, we need to call the RSS feed. If the cfhttp call times out, we know that the RSS feed is wrong. The URL could be invalid in other ways, but that's the only thing we check for in this code. XMLParse is called on cfhttp.filecontent. I wrote an article about getting and processing XML feeds this way, and you can read it at <http://coldfusion.sys-con.com/read/117667.htm>. The remainder of the code sets the instance variables and calls the commit. Congratulations, you've saved a feed in the database.

## What's Next

I've just about run out of space for this article, but this app is far from complete. In the next article I'll talk about the RSSAggregator and item components. It will create the scheduled task to call the aggregation code. Space permitting, I'll go into a gateway component along with some code that can be used to look at the aggregated data. If there isn't space in the next article for that, I'll fit it into some future article. Until next time, keep coding. 

## About the Author

Jeff Houser has been working with computers for over 20 years. He owns a DotComIt, a web consulting company, manages the CT Macromedia User Group, and routinely speaks and writes about development issues. You can find out what he's up to by checking his Blog at [www.jeffryhouser.com](http://www.jeffryhouser.com).

[jeff@instantcoldfusion.com](mailto:jeff@instantcoldfusion.com)