

[<< Back to Dashboard](#) | [<< Plugins Viewer](#)

Feed Reader

Introduction

The ColdBox feedReader plug-in is able to take any RSS, RDF or Atom feed and parse it into a standard CFML structure. This includes all feed dates which are automatically converted into CFML date strings. In addition the plug-in also supports file based caching using java serialization/de-serialization enabling your applications to save bandwidth where possible.

RSS Syndication Format

RSS as we know today originally started out back in March of 1999 as a RDF syndicated format for the My Netscape Network (MNN). MNN was operated by [Netscape](#) creators of the then popular and very famous Netscape web browser. By the next revision, the RDF syntax was dropped in favor of the current Really Simple Syndication syntax and has since evolved to become the most popular feed format.

RDF Syndication Format

RSS based on RDF was introduced by Netscape in the original RSS syndication format but by the second Netscape revision the RDF syntax was dropped. This led to the confusing situation where incompatible RDF and RSS syntax feeds were known by the same RSS acronym. In late 2000 an organization known as the RSS-DEV working group released RSS 1.0, but for simplicity we will refer to it as RDF 1.0. This feed format is based on the Resource Description Framework which is a WC3 Recommendation. It depends on modules such as the flexible Dublin Core Metadata Element Set to supply much of its feed data.

Atom Syndication Format

Atom was designed from the ground up as a vendor neutral alternative to the limited and confusing collection of RSS/RDF feed formats. Atom while very functional can be quite complex and so the adoption rate has been slower than expected. In December of 2005 the Atom Syndication Format was issued as a Proposed Standard in [IETF RFC 4287](#).

Supported Syntax and Formats

Feeds

- RDF : 0.90, 1.0
- RSS : 0.91, 0.92, 2.0, 2.0.x
- Atom : 0.3, 1.0

Extensions

- [A9 !OpenSearch 1 and 1.1 extensions](#)
- [Apple iTunes Podcast extension](#)
- [Content 2.0 RSS module](#)
- [Creative Commons RSS module](#)
- [Dublin Core Metadata Element Set 1.1](#)
- [Media RSS module](#)
- [Slash RSS module](#)

Resources

- [RSS Specification History](#)
- [RDF Site Summary \(RSS\)](#)
- [Atom Syndication Format - Introduction](#)

Basic Usage

While the feedReader plug-in has a number of methods, most of the time you will only ever need to use `readFeed(feedURL)` to retrieve feeds. This method will first look to see if there is a local, cache copy of the feed that has not expired. If that was not found it will retrieve the feed online using the URL provided by `feedURL`. The feed will be parsed and saved as a structure with the feed's items contained within a query.

```
<cfset rc.myFeed = getPlugin( "feedReader" ).readFeed( "http://www.example.com/rss" )>
```

A basic event to read and parse the [ColdBox Blog News RSS feed](#).

```
<--- Read & Parse the ColdBox Blog News Feed --->
<cffunction name="dumpFeed" access="public" returntype="void" output="false" >
  <cfargument name="Event" type="any" >
    <--- RC Reference --->
    <cfset var rc = event.getCollection() >
    <--- Obtain Feed --->
    <cfset rc.parsedFeed = getPlugin( "feedReader" ).readFeed( "http://blog.coldbox.org/feeds/rss.cfm" )>
    <--- Dump the Parsed Feed --->
    <cfset Event.setView( "vwFeedDump" )>
  </cffunction>
```

A simple view to dump the feed. Obviously in a production ColdBox application you would not dump the parsed feed, rather you would process the data returned.

```
<--- View to Display the Parsed ColdBox News Feed --->
<cfdump var="#rc.parsedFeed#" >
```

Configuring Your Application

The feedReader has a number of optional application settings that can be configured under the `YourSettings` element in your **ColdBox Configuration File**.

```
<YourSettings>
  <Setting name="feedReader_httpTimeout" value="60" />
  <Setting name="feedReader_cacheType" value="file" />
  <Setting name="feedReader_cacheLocation" value="temp/feedCache" />
</YourSettings>
```

Cache Settings

- `feedReader_useCache` : boolean [*default=true*] Cache and use a server stored copy of the feed
- `feedReader_cacheType` : string [*default=ram*] Store cache copy in server *ram* or to a server *file*
- `feedReader_cacheLocation` : string [*default=*] Relative or absolute path to a location to store file cache, this setting is ignored when `feedReader_cacheType=ram`
- `feedReader_cacheTimeout` : numeric [*default=30*] In minutes, the timeout of the feed cache

Other Settings

- `feedReader_httpTimeout` : numeric [*default=30*] In seconds, the timeout of the cfhttp (!ColdFusion HTTP request) call

Contents

- [Feed Reader](#)
 - [Introduction](#)
 - [RSS Syndication Format](#)
 - [RDF Syndication Format](#)
 - [Atom Syndication Format](#)
 - [Supported Syntax and Formats](#)
 - [Basic Usage](#)
 - [Configuring Your Application](#)
 - [Cache Settings](#)
 - [Other Settings](#)
 - [Advance Usage](#)
 - [Feed Retrieval Methods](#)
 - [Multiple Feeds Retrieval Methods](#)
 - [Cache Methods](#)
 - [Feed Parsed Output](#)
 - [Information Structure](#)
 - [OpenSearch Structure](#)
 - [Items as an Array](#)
 - [Items as a Query](#)

Advance Usage

In total there are 11 different methods that belong to the feedReader plug-in. Fortunately most of these relate to the control of the cache system and are usually not needed. All the methods can be found listed below, along with their purpose, syntax and a brief usage sample.

Feed Retrieval Methods

- **readFeed(*feedURL*, *itemsType*, *maxItems*)** This method parses a feed and returns the results. It first looks for a valid server cached copy of the feed before going online for a copy.


```
<cfset rc.myFeed = getPlugin( "feedReader" ).readFeed( "http://www.example.com/feeds/atom" , "array" ,20) >
```
- **retrieveFeed(*feedURL*, *itemsType*, *maxItems*)** This method parses a feed and returns the results. It only ever fetches the feed online and never uses a server cached copy.


```
<cfset rc.myFeed = getPlugin( "feedReader" ).retrieveFeed( "http://www.example.com/feeds/atom" , "query" ,0) >
```
- **parseFeed(*xmlDoc*, *itemsType*, *maxItems*)** This method parses a feed that is provided in a CFML variable. It can not use a cached copy or go online to fetch a feed.


```
<--- Read feed from a XML file stored on the server --->
<cfset rc.myFile = getPlugin( "Utilities" ).readFile( "C:\www\xml\rss.xml" ) >
<--- Apply the xmlParse() function on the rc.myFile variable and parse the feed --->
<cfset rc.myFeed = getPlugin( "feedReader" ).parseFeed(XmlParse(rc.myFile), "array" ,45) >
```

Arguments

- **feedURL** (*required*) URL to an online copy of the feed
- **xmlDoc** (*required*) Variable as a *XML document object* containing copy of the feed
- **itemsType** (*optional*) Parse the feed items as **query** or **array**, default is **query**
- **maxItems** (*optional*) Maximum number of feed items to display, default is **0** which displays all items

Multiple Feeds Retrieval Methods

If you wish to fetch more than a single feed and have it merged (or mashed) into a single master feed for display, then that is not a problem with ColdBox. Instead of supplying a single URL to the **feedURL** argument, you can supply a list of URLs. The feedReader will merge all the listed feeds together, reorder their items by the date published and output the results like a standard single feed. It doesn't even matter what format the feeds are, you can fetch and merge Atom, RSS and RDF feeds together.

In the simple example below 2 RSS and 1 Atom feed are being processed into a single feed. This feed will only display 10 times that will be in a query format.

```
<cfset rc.feed1 = "http://www.example.com/feeds/atom" >
<cfset rc.feed2 = "http://www.example.org/news/rss.xml" >
<cfset rc.feed3 = "http://www.example.org/siteupdates/rss.xml" >
<cfset rc.myFeed = getPlugin( "feedReader" ).retrieveFeed(feedURL= "#rc.feed1#,#rc.feed2#,#rc.feed3#" ,itemsType= "query" ,maxItems=10) >
```

The only down side to merging feeds is that some conflicting feed data will be dropped. The tags Author, Description, Image, Rating, Rights, OpenSearch will return empty values. Also obviously fetching and combining multiple feeds will require more processing time than a single feed.

Cache Methods

- **getCacheSize()** [*numeric*] Returns the number of feeds currently being cached, this does not count RAM cache
- **isFeedCached(*feedURL*)** [*boolean*] Checks to see if *feedURL* has a cached copy on the server
- **isFeedExpired(*feedURL*)** [*boolean*] Checks to see if the cache copy on the server belonging to *feedURL* has timed out
- **flushCache()** [*void*] Erases the entire feedReader cache, both RAM and files
- **expireCachedFeed(*feedURL*)** [*void*] Expire the cache belonging to *feedURL*
- **getCachedFeed(*feedURL*)** [*any*] Get the cache copy of *feedURL* and return it as a *XML document object*
- **removeCachedFeed(*feedURL*)** [*boolean*] Remove *feedURL* from the server cache
- **setCachedFeed(*feedURL*, *feedStruct*)** [*void*] Create a cache copy of *feedStruct* and use the url *feedURL* as the cache reference

```
<--- Example of flushing the cache --->
<cfscript>
if( getPlugin( "feedReader" ).getCacheSize() gte 1 ) {
  getPlugin( "feedReader" ).flushCache();
}
</cfscript>
```

Feed Parsed Output

Once the feed has been parsed, a structure will be created containing data extracted from the feed. You can then use this data for your own purposes.

* Starred structures/arrays will return an empty state if the result is empty. So if there are no categories, the **Category** array will return an empty array. Rather than returning a single array with empty **Domain** and **Tag** structures values.

Information Structure

The following parsed structure keys pertain to information about the feed. All values are strings unless otherwise noted and are only ever displayed once.

```
Author [structure ]::
  Email E-mail address of the primary author
  Name Name of the primary author
  Url URL link to the primary author
Category* [array ]::
  Domain URL to or name of the convention used by this category
  Tag Category item
Datebuilt When feed was last compiled::
Dateupdated When feed data was last updated::
Description Text about this feed::
Image [structure ]::
  Description Text about this logo
  Height Logo height in pixels
  Icon URL to a tiny version of this logo
  Link URL of a website to link to when logo is clicked
  Title Title for logo
  Url URL of where logo is stored
  Width Logo width in pixels
Language Primary feed language, usually a HTML language code::
Rating Content notifications and warnings such as offensive language::
Rights [structure ]::
  Copyright Copyright notice
  Creativecommons URL pointing to the feed's Creative Commons license
Title Title of the feed::
Websiteurl URL of the website this feed is associated with::
```

OpenSearch Structure

OpenSearch is a collection of technologies that (depending on the implementation) allow you to interact with website, search engine results.

```
Opensearch* [structure ]::
  Autodiscovery [structure ]::
    Title Title for the !OpenSearch description document
    Url URL to the !OpenSearch description document
```

```

Itemsperpage Total number of results from the !OpenSearch request contained in this feed
Startindex Current page of the !OpenSearch request
Totalresults Total number of results from the !OpenSearch request
Query {array}::
  Role Identify what this !OpenSearch query is used for
  Totalresults Expected number of results
  Searchterms Search terms requested
  Count Search results per page
  Startindex Search index number of the first search request
  Startpage Search page number of the first search request
  Language Search results language
  Inputencoding Search request character encoding
  Outputencoding Search results character encoding

```

Items as an Array

The following structure keys contain item data. Each item is stored as a query or by default as an array. Query results don't return anywhere near the same amount of information as the array. The following is the item structure returned as an array, all values are strings unless otherwise noted.

```

Items {array}::
Attachment* {array}::
  See below
Author Author of the item or file attachment::
Body Text describing this feed, the attachment or a text summary of a longer article::
Category* {array}::
  Domain URL to or name of the convention used by this category
  Tag Category item
Comments {structure}::
  Count Number of comments for this item
  Hit_parade A history list of the comment count
  Url A URL to a webpage where you can leave comments in regards to this item
Datepublished When item was first published::
Dateupdated When item was last updated::
Id (Should be) a unique Id for this item::
Keywords A list of classifications used to reference the item, these are sourced from the category array::
Rights Copyright notice for this item::
Title Title for this item::
Url URL to the HTML or complete edition of this item::

```

The **Attachment** array can have three different structures dependent on the source. You can use the **type** value to detect what kind of array structure will be returned.

```

Attachment* {array}::
  Type "enclosure" A standard feed file attachment
  Duration Running duration or length
  Filesize File size (usually in bytes)
  Mimetype MIME type
  Url URL of where the file is located so it can be directly downloaded

Attachment* {array}::
  Type "media" A Media RSS attachment
  Duration Play time in seconds
  Filesize File size in bytes
  Height Media height
  ISDefault When the value is true then this is the default attachment for use
  Medium Type of media, value will be either image, audio, video, document, executable
  Mimetype MIME type
  Url URL of where the media is located so it can be directly downloaded or streamed
  Width Media width

Attachment* {array}::
  Type "thumbnail" A Media RSS thumbnail preview
  Height Thumbnail height
  Medium Will always be image
  Url URL of where the thumbnail is located so it can be directly downloaded or viewed
  Width Thumbnail width

```

The **Specs** structure contains technical information related to the feed, it can usually be ignored.

```

Specs {structure}::
Extensions List of recognised extensions contained within this feed
Generator Name (and maybe version, URL) of the program used to create this feed
Namespace {structure}
  Namespace structure contains a collection of URLs returned by the feed's namespace tag
Type Type of feed, either RSS, RDF, Atom
Url Self URL for this feed (this is supplied by the feed and is not generated by ColdBox)
Version Numeric revision value for the feed, best used in combination with Type

```

Items as a Query

The following structure keys contain limited item data as returned by a query with string values.

```

Items {query}::
Attachment URL linking to the first file attachment within the item::
Author Author of the item or file attachment::
Body Text describing this feed, the attachment or a text summary of a longer article::
Category A list of classifications used to reference the item::
Comments Number of comments for this item::
Datepublished When item was first published::
Dateupdated When item was last updated::
Id (Should be) a unique Id for this item::
Rights Copyright notice for this item::
Title Title for this item::
Url URL to the HTML or complete edition of this item::

```

Categories:

- [level-intermediate](#)