C# Reference Classes for Reading RSS and Atom Feeds

Dario Airoldi

2025-10-25

Table of contents

1 C# Reference Classes for Reading RSS and Atom Feeds

Complete, production-ready C# class definitions for parsing both RSS 2.0 and Atom feeds, implementing the full specifications with iTunes podcast extensions and WebSub support.

1.1 Table of Contents

1.2 Introduction

This document provides **complete, production-ready C# class definitions** for parsing both RSS 2.0 and Atom syndication feeds. The implementation includes:

- **Full RSS 2.0 support** with iTunes podcast extensions
- **Complete Atom (RFC 4287) implementation** with all constructs
- **WebSub discovery** for both RSS and Atom feeds
- **Type-safe** strongly-typed properties
- **Extensible architecture** using abstract base classes
- **Date parsing** for RFC 822 (RSS) and RFC 3339 (Atom) formats

- **Media enclosure** support for podcasts and multimedia

1.2.1 Design Principles

1. **Standards Compliance**: Implements RSS 2.0 and Atom (RFC 4287) specifications completely
2. **Type Safety**: Uses nullable types for optional fields, enums for fixed values
3. **Extensibility**: Abstract base classes allow custom implementations
4. **Simplicity**: Clean, readable code with comprehensive documentation
5. **Reusability**: Modular design suitable for any .NET application

---

1.3 Base Classes

1.3.1 FeedChannelBase - Abstract Base for Feed Metadata

The abstract base class for feed-level metadata (channel/feed), providing common properties shared by both RSS and Atom feeds.

```
using System;
using System.Collections.Generic;

/// <summary>
/// Abstract base class for feed-level metadata (channel/feed)
/// </summary>
public abstract class FeedChannelBase
{
    /// <summary>
    /// Unique identifier for the feed
    /// </summary>
    public string Id { get; set; }

    /// <summary>
    /// Human-readable feed title
    /// </summary>
    public string Title { get; set; }

    /// <summary>
    /// Feed description or subtitle
    /// </summary>
```

```
    public string Description { get; set; }

    /// <summary>
    /// Website URL associated with the feed
    /// </summary>
    public string Link { get; set; }

    /// <summary>
    /// Language code (e.g., "en-US", "fr-FR")
    /// </summary>
    public string Language { get; set; }

    /// <summary>
    /// Copyright/rights information
    /// </summary>
    public string Copyright { get; set; }

    /// <summary>
    /// Last update/modification timestamp
    /// </summary>
    public DateTime? LastUpdated { get; set; }

    /// <summary>
    /// Publication date
    /// </summary>
    public DateTime? PublicationDate { get; set; }

    /// <summary>
    /// Feed categories/tags
    /// </summary>
    public List<string> Categories { get; set; } = new List<string>();

    /// <summary>
    /// Feed image/logo URL
    /// </summary>
    public string ImageUrl { get; set; }

    /// <summary>
    /// Software that generated the feed
    /// </summary>
    public string Generator { get; set; }
```

```
    /// <summary>
    /// Collection of feed items/entries
    /// </summary>
    public List<FeedItemBase> Items { get; set; } = new List<FeedItemBase>();

    /// <summary>
    /// Feed format type
    /// </summary>
    public abstract FeedType FeedType { get; }
}

/// <summary>
/// Enumeration of supported feed types
/// </summary>
public enum FeedType
{
    /// <summary>
    /// RSS 2.0 feed format
    /// </summary>
    RSS20,

    /// <summary>
    /// Atom (RFC 4287) feed format
    /// </summary>
    Atom
}
```

1.3.2 FeedItemBase - Abstract Base for Feed Entries

The abstract base class for individual feed items/entries, providing common properties for both RSS items and Atom entries.

```
using System;
using System.Collections.Generic;

/// <summary>
/// Abstract base class for individual feed items/entries
/// </summary>
public abstract class FeedItemBase
```

```
{
    /// <summary>
    /// Unique identifier for the item
    /// </summary>
    public string Id { get; set; }

    /// <summary>
    /// Item title
    /// </summary>
    public string Title { get; set; }

    /// <summary>
    /// Item description or summary
    /// </summary>
    public string Description { get; set; }

    /// <summary>
    /// Permanent URL for the item
    /// </summary>
    public string Link { get; set; }

    /// <summary>
    /// Author information
    /// </summary>
    public string Author { get; set; }

    /// <summary>
    /// Publication date
    /// </summary>
    public DateTime? PublicationDate { get; set; }

    /// <summary>
    /// Last update/modification timestamp
    /// </summary>
    public DateTime? LastUpdated { get; set; }

    /// <summary>
    /// Item categories/tags
    /// </summary>
    public List<string> Categories { get; set; } = new List<string>();

    /// <summary>
```

```
    /// Media enclosures (audio, video, attachments)
    /// </summary>
    public List<MediaEnclosure> Enclosures { get; set; } = new List<MediaEnclosure>();

    /// <summary>
    /// Item format type
    /// </summary>
    public abstract FeedType ItemType { get; }
}
```

1.3.3 MediaEnclosure - Podcast and Media Attachments

Represents a media enclosure (podcast audio, video, or other attachments) with size and type information.

```
using System;

/// <summary>
/// Represents a media enclosure (podcast audio, video, etc.)
/// </summary>
public class MediaEnclosure
{
    /// <summary>
    /// Direct URL to the media file
    /// </summary>
    public string Url { get; set; }

    /// <summary>
    /// MIME type (e.g., "audio/mpeg", "video/mp4")
    /// </summary>
    public string Type { get; set; }

    /// <summary>
    /// File size in bytes
    /// </summary>
    public long? Length { get; set; }

    /// <summary>
    /// Media duration (for audio/video)
```

```
    /// </summary>
    public TimeSpan? Duration { get; set; }
}
```

1.4 RSS 2.0 Implementation

1.4.1 RSSFeedChannel - RSS Channel with iTunes Extensions

Complete RSS 2.0 channel implementation with full support for iTunes podcast extensions and WebSub.

```
using System;
using System.Collections.Generic;

/// <summary>
/// RSS 2.0 channel/feed implementation with iTunes extensions
/// </summary>
public class RSSFeedChannel : FeedChannelBase
{
    /// <summary>
    /// Managing editor email address
    /// </summary>
    public string ManagingEditor { get; set; }

    /// <summary>
    /// Webmaster email address
    /// </summary>
    public string WebMaster { get; set; }

    /// <summary>
    /// Time-to-live in minutes (caching hint)
    /// </summary>
    public int? Ttl { get; set; }

    /// <summary>
    /// Hours when aggregators should skip updates (0-23)
    /// </summary>
    public List<int> SkipHours { get; set; } = new List<int>();
```

```
    /// <summary>
    /// Days when aggregators should skip updates
    /// </summary>
    public List<string> SkipDays { get; set; } = new List<string>();

    /// <summary>
    /// URL to RSS specification documentation
    /// </summary>
    public string Docs { get; set; }

    /// <summary>
    /// Cloud notification settings (RSSCloud)
    /// </summary>
    public CloudSettings Cloud { get; set; }

    /// <summary>
    /// WebSub hub URL for push notifications
    /// </summary>
    public string WebSubHub { get; set; }

    /// <summary>
    /// Self-reference URL (for WebSub)
    /// </summary>
    public string SelfLink { get; set; }

    // iTunes Podcast Extensions

    /// <summary>
    /// iTunes podcast author
    /// </summary>
    public string ItunesAuthor { get; set; }

    /// <summary>
    /// iTunes podcast subtitle
    /// </summary>
    public string ItunesSubtitle { get; set; }

    /// <summary>
    /// iTunes podcast summary
    /// </summary>
    public string ItunesSummary { get; set; }
```

```
    /// <summary>
    /// iTunes explicit content rating
    /// </summary>
    public bool? ItunesExplicit { get; set; }

    /// <summary>
    /// iTunes podcast type (episodic or serial)
    /// </summary>
    public string ItunesType { get; set; }

    /// <summary>
    /// iTunes owner information
    /// </summary>
    public ItunesOwner ItunesOwner { get; set; }

    /// <summary>
    /// iTunes artwork URL
    /// </summary>
    public string ItunesImageUrl { get; set; }

    /// <summary>
    /// iTunes categories
    /// </summary>
    public List<ItunesCategory> ItunesCategories { get; set; } = new List<ItunesCategory>();

    /// <summary>
    /// Feed type identifier
    /// </summary>
    public override FeedType FeedType => FeedType.RSS20;
}
```

1.4.2 Supporting RSS Classes

1.4.2.1 CloudSettings - RSSCloud Notification Configuration

```
/// <summary>
/// RSS Cloud notification settings (RSSCloud protocol)
/// </summary>
public class CloudSettings
```

```
{
    /// <summary>
    /// Cloud server domain
    /// </summary>
    public string Domain { get; set; }

    /// <summary>
    /// Cloud server port
    /// </summary>
    public int Port { get; set; }

    /// <summary>
    /// Cloud server path
    /// </summary>
    public string Path { get; set; }

    /// <summary>
    /// Registration procedure name
    /// </summary>
    public string RegisterProcedure { get; set; }

    /// <summary>
    /// Notification protocol (xml-rpc, soap, http-post)
    /// </summary>
    public string Protocol { get; set; }
}
```

1.4.2.2 ItunesOwner - Podcast Owner Information

```
/// <summary>
/// iTunes podcast owner information
/// </summary>
public class ItunesOwner
{
    /// <summary>
    /// Owner name
    /// </summary>
    public string Name { get; set; }

    /// <summary>
    /// Owner email address
```

```
    /// </summary>
    public string Email { get; set; }
}
```

1.4.2.3 ItunesCategory - Podcast Category Structure

```
/// <summary>
/// iTunes category structure with optional subcategory
/// </summary>
public class ItunesCategory
{
    /// <summary>
    /// Category text/label
    /// </summary>
    public string Text { get; set; }

    /// <summary>
    /// Optional subcategory
    /// </summary>
    public ItunesCategory Subcategory { get; set; }
}
```

1.4.3 RSSFeedItem - RSS Item with iTunes Extensions

Complete RSS 2.0 item implementation with full iTunes podcast episode extensions.

```
using System;

/// <summary>
/// RSS 2.0 item implementation with iTunes extensions
/// </summary>
public class RSSFeedItem : FeedItemBase
{
    /// <summary>
    /// GUID (Globally Unique Identifier)
    /// </summary>
    public string Guid { get; set; }
```

```
    /// <summary>
    /// Whether GUID is a permalink
    /// </summary>
    public bool GuidIsPermaLink { get; set; } = true;

    /// <summary>
    /// URL to comments page
    /// </summary>
    public string Comments { get; set; }

    /// <summary>
    /// Source feed information (for aggregated content)
    /// </summary>
    public RSSSource Source { get; set; }

    // iTunes Episode Extensions

    /// <summary>
    /// iTunes episode author
    /// </summary>
    public string ItunesAuthor { get; set; }

    /// <summary>
    /// iTunes episode subtitle
    /// </summary>
    public string ItunesSubtitle { get; set; }

    /// <summary>
    /// iTunes episode summary
    /// </summary>
    public string ItunesSummary { get; set; }

    /// <summary>
    /// iTunes episode duration
    /// </summary>
    public TimeSpan? ItunesDuration { get; set; }

    /// <summary>
    /// iTunes episode explicit rating
    /// </summary>
    public bool? ItunesExplicit { get; set; }
```

```
    /// <summary>
    /// iTunes episode artwork URL
    /// </summary>
    public string ItunesImageUrl { get; set; }

    /// <summary>
    /// iTunes episode number
    /// </summary>
    public int? ItunesEpisode { get; set; }

    /// <summary>
    /// iTunes season number
    /// </summary>
    public int? ItunesSeason { get; set; }

    /// <summary>
    /// iTunes episode type (full, trailer, bonus)
    /// </summary>
    public string ItunesEpisodeType { get; set; }

    /// <summary>
    /// Item type identifier
    /// </summary>
    public override FeedType ItemType => FeedType.RSS20;
}
```

1.4.3.1 RSSSource - Source Feed Reference

```
/// <summary>
/// RSS source information for republished content
/// </summary>
public class RSSSource
{
    /// <summary>
    /// Source feed URL
    /// </summary>
    public string Url { get; set; }

    /// <summary>
    /// Source feed title
    /// </summary>
```

```
    public string Title { get; set; }
}
```

1.5 Atom Implementation

1.5.1 AtomFeedChannel - Atom Feed

Complete Atom (RFC 4287) feed implementation with all constructs and WebSub support.

```
using System;
using System.Collections.Generic;
using System.Linq;

/// <summary>
/// Atom syndication feed implementation (RFC 4287)
/// </summary>
public class AtomFeedChannel : FeedChannelBase
{
    /// <summary>
    /// Feed subtitle/tagline
    /// </summary>
    public string Subtitle { get; set; }

    /// <summary>
    /// Feed authors
    /// </summary>
    public List<AtomPerson> Authors { get; set; } = new List<AtomPerson>();

    /// <summary>
    /// Feed contributors
    /// </summary>
    public List<AtomPerson> Contributors { get; set; } = new List<AtomPerson>();

    /// <summary>
    /// Feed links (alternate, self, related, hub)
    /// </summary>
    public List<AtomLink> Links { get; set; } = new List<AtomLink>();

    /// <summary>
```

```
    /// Feed categories with schemes
    /// </summary>
    public new List<AtomCategory> Categories { get; set; } = new List<AtomCategory>();

    /// <summary>
    /// Feed icon URL (small, 1:1 aspect ratio)
    /// </summary>
    public string Icon { get; set; }

    /// <summary>
    /// Feed logo URL (larger, 2:1 aspect ratio)
    /// </summary>
    public string Logo { get; set; }

    /// <summary>
    /// Generator information
    /// </summary>
    public AtomGenerator Generator { get; set; }

    /// <summary>
    /// WebSub hub URL (from link rel="hub")
    /// </summary>
    public string WebSubHub
    {
        get => Links?.Find(l => l.Relation == "hub")?.Href;
    }

    /// <summary>
    /// Self-reference URL (from link rel="self")
    /// </summary>
    public string SelfLink
    {
        get => Links?.Find(l => l.Relation == "self")?.Href;
    }

    /// <summary>
    /// Feed type identifier
    /// </summary>
    public override FeedType FeedType => FeedType.Atom;
}
```

1.5.2 Atom Constructs

1.5.2.1 AtomPerson - Author and Contributor

```
/// <summary>
/// Atom person construct (author, contributor)
/// </summary>
public class AtomPerson
{
    /// <summary>
    /// Person's name (required)
    /// </summary>
    public string Name { get; set; }

    /// <summary>
    /// IRI associated with person (homepage, profile)
    /// </summary>
    public string Uri { get; set; }

    /// <summary>
    /// Email address
    /// </summary>
    public string Email { get; set; }

    /// <summary>
    /// Returns the person's name
    /// </summary>
    public override string ToString() => Name;
}
```

1.5.2.2 AtomLink - Link Relationships

```
/// <summary>
/// Atom link construct with relationship types
/// </summary>
public class AtomLink
{
    /// <summary>
    /// IRI reference (required)
    /// </summary>
    public string Href { get; set; }
```

```
    /// <summary>
    /// Link relationship type (alternate, enclosure, self, related, via, hub)
    /// </summary>
    public string Relation { get; set; } = "alternate";

    /// <summary>
    /// MIME media type
    /// </summary>
    public string Type { get; set; }

    /// <summary>
    /// Language of linked resource
    /// </summary>
    public string HrefLang { get; set; }

    /// <summary>
    /// Human-readable title
    /// </summary>
    public string Title { get; set; }

    /// <summary>
    /// Size in bytes (for enclosures)
    /// </summary>
    public long? Length { get; set; }
}
```

1.5.2.3 AtomCategory - Categorization

```
/// <summary>
/// Atom category construct
/// </summary>
public class AtomCategory
{
    /// <summary>
    /// Category identifier (required)
    /// </summary>
    public string Term { get; set; }

    /// <summary>
    /// Categorization scheme IRI
    /// </summary>
```

```
    public string Scheme { get; set; }

    /// <summary>
    /// Human-readable label
    /// </summary>
    public string Label { get; set; }

    /// <summary>
    /// Returns the label or term
    /// </summary>
    public override string ToString() => Label ?? Term;
}
```

1.5.2.4 AtomGenerator - Feed Generator Information

```
/// <summary>
/// Atom generator information
/// </summary>
public class AtomGenerator
{
    /// <summary>
    /// Generator name/text
    /// </summary>
    public string Text { get; set; }

    /// <summary>
    /// Generator URI
    /// </summary>
    public string Uri { get; set; }

    /// <summary>
    /// Generator version
    /// </summary>
    public string Version { get; set; }

    /// <summary>
    /// Returns formatted generator information
    /// </summary>
    public override string ToString() =>
        string.IsNullOrEmpty(Version) ? Text : $"{Text} {Version}";
}
```

1.5.3 AtomFeedItem - Atom Entry

Complete Atom entry implementation with all optional elements.

```
using System;
using System.Collections.Generic;

/// <summary>
/// Atom entry implementation (RFC 4287)
/// </summary>
public class AtomFeedItem : FeedItemBase
{
    /// <summary>
    /// Entry authors
    /// </summary>
    public List<AtomPerson> Authors { get; set; } = new List<AtomPerson>();

    /// <summary>
    /// Entry contributors
    /// </summary>
    public List<AtomPerson> Contributors { get; set; } = new List<AtomPerson>();

    /// <summary>
    /// Entry links (alternate, enclosure, related)
    /// </summary>
    public List<AtomLink> Links { get; set; } = new List<AtomLink>();

    /// <summary>
    /// Entry categories with schemes
    /// </summary>
    public new List<AtomCategory> Categories { get; set; } = new List<AtomCategory>();

    /// <summary>
    /// Entry content
    /// </summary>
    public AtomContent Content { get; set; }

    /// <summary>
    /// Entry summary/excerpt
    /// </summary>
```

```
    public AtomText Summary { get; set; }

    /// <summary>
    /// Original publication timestamp
    /// </summary>
    public DateTime? Published { get; set; }

    /// <summary>
    /// Rights/copyright information
    /// </summary>
    public string Rights { get; set; }

    /// <summary>
    /// Source feed metadata (for aggregated entries)
    /// </summary>
    public AtomSource Source { get; set; }

    /// <summary>
    /// Item type identifier
    /// </summary>
    public override FeedType ItemType => FeedType.Atom;
}
```

1.5.3.1 AtomContent - Entry Content

```
/// <summary>
/// Atom content construct
/// </summary>
public class AtomContent
{
    /// <summary>
    /// Content type (text, html, xhtml, or MIME type)
    /// </summary>
    public string Type { get; set; } = "text";

    /// <summary>
    /// Content text/markup
    /// </summary>
    public string Text { get; set; }

    /// <summary>
```

```
    /// External content IRI (for out-of-line content)
    /// </summary>
    public string Src { get; set; }

    /// <summary>
    /// Whether content is inline or external
    /// </summary>
    public bool IsInline => string.IsNullOrEmpty(Src);
}
```

1.5.3.2 AtomText - Text Construct

```
/// <summary>
/// Atom text construct (for summary, title, etc.)
/// </summary>
public class AtomText
{
    /// <summary>
    /// Text type (text, html, xhtml)
    /// </summary>
    public string Type { get; set; } = "text";

    /// <summary>
    /// Text content
    /// </summary>
    public string Text { get; set; }

    /// <summary>
    /// Returns the text content
    /// </summary>
    public override string ToString() => Text;
}
```

1.5.3.3 AtomSource - Source Feed Metadata

```
using System;
using System.Collections.Generic;

/// <summary>
/// Atom source metadata for aggregated entries
```

```
/// </summary>
public class AtomSource
{
    /// <summary>
    /// Source feed ID
    /// </summary>
    public string Id { get; set; }

    /// <summary>
    /// Source feed title
    /// </summary>
    public string Title { get; set; }

    /// <summary>
    /// Source feed last update time
    /// </summary>
    public DateTime? Updated { get; set; }

    /// <summary>
    /// Source feed links
    /// </summary>
    public List<AtomLink> Links { get; set; } = new List<AtomLink>();
}
```

1.6 Feed Parsers

1.6.1 RSSFeedParser - Parse RSS 2.0 Feeds

Complete RSS parser with iTunes extensions and WebSub discovery.

```
using System;
using System.Linq;
using System.Xml.Linq;

/// <summary>
/// Parser for RSS 2.0 feeds with iTunes extensions
/// </summary>
public class RSSFeedParser
{
```

```
    /// <summary>
    /// Parse RSS 2.0 feed from XML string
    /// </summary>
    /// <param name="xmlContent">RSS XML content</param>
    /// <returns>Parsed RSS feed channel</returns>
    public static RSSFeedChannel ParseRSS(string xmlContent)
    {
        var doc = XDocument.Parse(xmlContent);
        var channel = doc.Descendants("channel").FirstOrDefault();

        if (channel == null)
            throw new InvalidOperationException("Invalid RSS feed: <channel> element not fou

        var itunesNs = XNamespace.Get("http://www.itunes.com/dtds/podcast-1.0.dtd");
        var atomNs = XNamespace.Get("http://www.w3.org/2005/Atom");

        var rssFeed = new RSSFeedChannel
        {
            Id = channel.Element("link")?.Value,
            Title = channel.Element("title")?.Value,
            Description = channel.Element("description")?.Value,
            Link = channel.Element("link")?.Value,
            Language = channel.Element("language")?.Value,
            Copyright = channel.Element("copyright")?.Value,
            ManagingEditor = channel.Element("managingEditor")?.Value,
            WebMaster = channel.Element("webMaster")?.Value,
            Generator = channel.Element("generator")?.Value,
            ImageUrl = channel.Element("image")?.Element("url")?.Value,

            // Parse dates
            PublicationDate = ParseRFC822Date(channel.Element("pubDate")?.Value),
            LastUpdated = ParseRFC822Date(channel.Element("lastBuildDate")?.Value),

            // Parse TTL
            Ttl = int.TryParse(channel.Element("ttl")?.Value, out int ttl) ? ttl : (int?)null

            // WebSub support
            WebSubHub = channel.Elements(atomNs + "link")
                .FirstOrDefault(l => (string)l.Attribute("rel") == "hub")?
                .Attribute("href")?.Value,
            SelfLink = channel.Elements(atomNs + "link")
                .FirstOrDefault(l => (string)l.Attribute("rel") == "self")?
```

```
                .Attribute("href")?.Value,

            // iTunes extensions
            ItunesAuthor = channel.Element(itunesNs + "author")?.Value,
            ItunesSubtitle = channel.Element(itunesNs + "subtitle")?.Value,
            ItunesSummary = channel.Element(itunesNs + "summary")?.Value,
            ItunesImageUrl = channel.Element(itunesNs + "image")?.Attribute("href")?.Value,
            ItunesExplicit = ParseItunesExplicit(channel.Element(itunesNs + "explicit")?.Valu
            ItunesType = channel.Element(itunesNs + "type")?.Value
        };

        // Parse categories
        rssFeed.Categories.AddRange(
            channel.Elements("category").Select(c => c.Value)
        );

        // Parse items
        foreach (var item in channel.Elements("item"))
        {
            rssFeed.Items.Add(ParseRSSItem(item, itunesNs));
        }

        return rssFeed;
    }

    private static RSSFeedItem ParseRSSItem(XElement item, XNamespace itunesNs)
    {
        var rssItem = new RSSFeedItem
        {
            Title = item.Element("title")?.Value,
            Description = item.Element("description")?.Value,
            Link = item.Element("link")?.Value,
            Author = item.Element("author")?.Value,
            Comments = item.Element("comments")?.Value,

            // GUID
            Guid = item.Element("guid")?.Value,
            GuidIsPermaLink = item.Element("guid")?.Attribute("isPermaLink")?.Value != "false

            // Dates
            PublicationDate = ParseRFC822Date(item.Element("pubDate")?.Value),
```

```
            // iTunes extensions
            ItunesAuthor = item.Element(itunesNs + "author")?.Value,
            ItunesSubtitle = item.Element(itunesNs + "subtitle")?.Value,
            ItunesSummary = item.Element(itunesNs + "summary")?.Value,
            ItunesImageUrl = item.Element(itunesNs + "image")?.Attribute("href")?.Value,
            ItunesExplicit = ParseItunesExplicit(item.Element(itunesNs + "explicit")?.Value)
            ItunesDuration = ParseItunesDuration(item.Element(itunesNs + "duration")?.Value)
            ItunesEpisode = int.TryParse(item.Element(itunesNs + "episode")?.Value, out int e
            ItunesSeason = int.TryParse(item.Element(itunesNs + "season")?.Value, out int sea
            ItunesEpisodeType = item.Element(itunesNs + "episodeType")?.Value
        };

        // Set ID
        rssItem.Id = rssItem.Guid ?? rssItem.Link;

        // Parse categories
        rssItem.Categories.AddRange(
            item.Elements("category").Select(c => c.Value)
        );

        // Parse enclosures
        foreach (var enc in item.Elements("enclosure"))
        {
            rssItem.Enclosures.Add(new MediaEnclosure
            {
                Url = enc.Attribute("url")?.Value,
                Type = enc.Attribute("type")?.Value,
                Length = long.TryParse(enc.Attribute("length")?.Value, out long len) ? len :
                Duration = rssItem.ItunesDuration
            });
        }

        return rssItem;
    }

    private static DateTime? ParseRFC822Date(string dateString)
    {
        if (string.IsNullOrWhiteSpace(dateString))
            return null;

        try
        {
```

```
            return DateTime.Parse(dateString,
                System.Globalization.CultureInfo.InvariantCulture,
                System.Globalization.DateTimeStyles.AdjustToUniversal);
        }
        catch
        {
            return null;
        }
    }

    private static bool? ParseItunesExplicit(string value)
    {
        if (string.IsNullOrWhiteSpace(value))
            return null;

        return value.ToLower() == "true" || value.ToLower() == "yes";
    }

    private static TimeSpan? ParseItunesDuration(string duration)
    {
        if (string.IsNullOrWhiteSpace(duration))
            return null;

        // Format: HH:MM:SS or MM:SS or seconds
        if (TimeSpan.TryParse(duration, out TimeSpan ts))
            return ts;

        if (int.TryParse(duration, out int seconds))
            return TimeSpan.FromSeconds(seconds);

        return null;
    }
}
```

1.6.2 AtomFeedParser - Parse Atom Feeds

Complete Atom parser implementing RFC 4287 with WebSub discovery.

```
using System;
using System.Linq;
using System.Xml.Linq;

/// <summary>
/// Parser for Atom (RFC 4287) feeds
/// </summary>
public class AtomFeedParser
{
    /// <summary>
    /// Parse Atom feed from XML string
    /// </summary>
    /// <param name="xmlContent">Atom XML content</param>
    /// <returns>Parsed Atom feed</returns>
    public static AtomFeedChannel ParseAtom(string xmlContent)
    {
        var doc = XDocument.Parse(xmlContent);
        var ns = doc.Root.GetDefaultNamespace();
        var feed = doc.Root;

        if (feed.Name.LocalName != "feed")
            throw new InvalidOperationException("Invalid Atom feed: <feed> element not found

        var atomFeed = new AtomFeedChannel
        {
            Id = feed.Element(ns + "id")?.Value,
            Title = feed.Element(ns + "title")?.Value,
            Description = feed.Element(ns + "subtitle")?.Value,
            Subtitle = feed.Element(ns + "subtitle")?.Value,
            Copyright = feed.Element(ns + "rights")?.Value,
            Icon = feed.Element(ns + "icon")?.Value,
            Logo = feed.Element(ns + "logo")?.Value,

            // Dates
            LastUpdated = ParseRFC3339Date(feed.Element(ns + "updated")?.Value)
        };

        // Parse generator
        var genElement = feed.Element(ns + "generator");
        if (genElement != null)
        {
            atomFeed.Generator = new AtomGenerator
```

```
            {
                Text = genElement.Value,
                Uri = genElement.Attribute("uri")?.Value,
                Version = genElement.Attribute("version")?.Value
            };
            atomFeed.Generator = atomFeed.Generator.Text;
        }

        // Parse authors
        foreach (var author in feed.Elements(ns + "author"))
        {
            atomFeed.Authors.Add(ParseAtomPerson(author, ns));
        }

        // Parse contributors
        foreach (var contributor in feed.Elements(ns + "contributor"))
        {
            atomFeed.Contributors.Add(ParseAtomPerson(contributor, ns));
        }

        // Parse links
        foreach (var link in feed.Elements(ns + "link"))
        {
            atomFeed.Links.Add(ParseAtomLink(link));
        }

        // Set main link (alternate)
        atomFeed.Link = atomFeed.Links
            .FirstOrDefault(l => l.Relation == "alternate" && l.Type == "text/html")?
            .Href;

        // Parse categories
        foreach (var cat in feed.Elements(ns + "category"))
        {
            atomFeed.Categories.Add(ParseAtomCategory(cat));
        }

        // Parse entries
        foreach (var entry in feed.Elements(ns + "entry"))
        {
            atomFeed.Items.Add(ParseAtomEntry(entry, ns));
        }
```

```
        return atomFeed;
    }

    private static AtomFeedItem ParseAtomEntry(XElement entry, XNamespace ns)
    {
        var atomItem = new AtomFeedItem
        {
            Id = entry.Element(ns + "id")?.Value,
            Title = entry.Element(ns + "title")?.Value,
            Rights = entry.Element(ns + "rights")?.Value,

            // Dates
            LastUpdated = ParseRFC3339Date(entry.Element(ns + "updated")?.Value),
            Published = ParseRFC3339Date(entry.Element(ns + "published")?.Value)
        };

        // Use published date as publication date
        atomItem.PublicationDate = atomItem.Published ?? atomItem.LastUpdated;

        // Parse summary
        var summaryElement = entry.Element(ns + "summary");
        if (summaryElement != null)
        {
            atomItem.Summary = new AtomText
            {
                Type = summaryElement.Attribute("type")?.Value ?? "text",
                Text = summaryElement.Value
            };
            atomItem.Description = atomItem.Summary.Text;
        }

        // Parse content
        var contentElement = entry.Element(ns + "content");
        if (contentElement != null)
        {
            atomItem.Content = new AtomContent
            {
                Type = contentElement.Attribute("type")?.Value ?? "text",
                Src = contentElement.Attribute("src")?.Value,
                Text = contentElement.Value
            };
        }
```

```
        // Parse authors
        foreach (var author in entry.Elements(ns + "author"))
        {
            atomItem.Authors.Add(ParseAtomPerson(author, ns));
        }

        // Set author string
        atomItem.Author = string.Join(", ", atomItem.Authors.Select(a => a.Name));

        // Parse contributors
        foreach (var contributor in entry.Elements(ns + "contributor"))
        {
            atomItem.Contributors.Add(ParseAtomPerson(contributor, ns));
        }

        // Parse links
        foreach (var link in entry.Elements(ns + "link"))
        {
            var atomLink = ParseAtomLink(link);
            atomItem.Links.Add(atomLink);

            // Handle enclosures
            if (atomLink.Relation == "enclosure")
            {
                atomItem.Enclosures.Add(new MediaEnclosure
                {
                    Url = atomLink.Href,
                    Type = atomLink.Type,
                    Length = atomLink.Length
                });
            }
        }

        // Set main link (alternate)
        atomItem.Link = atomItem.Links
            .FirstOrDefault(l => l.Relation == "alternate")?
            .Href;

        // Parse categories
        foreach (var cat in entry.Elements(ns + "category"))
        {
            atomItem.Categories.Add(ParseAtomCategory(cat));
```

```
        }

        return atomItem;
    }

    private static AtomPerson ParseAtomPerson(XElement person, XNamespace ns)
    {
        return new AtomPerson
        {
            Name = person.Element(ns + "name")?.Value,
            Uri = person.Element(ns + "uri")?.Value,
            Email = person.Element(ns + "email")?.Value
        };
    }

    private static AtomLink ParseAtomLink(XElement link)
    {
        return new AtomLink
        {
            Href = link.Attribute("href")?.Value,
            Relation = link.Attribute("rel")?.Value ?? "alternate",
            Type = link.Attribute("type")?.Value,
            HrefLang = link.Attribute("hreflang")?.Value,
            Title = link.Attribute("title")?.Value,
            Length = long.TryParse(link.Attribute("length")?.Value, out long len) ? len : (lo
        };
    }

    private static AtomCategory ParseAtomCategory(XElement category)
    {
        return new AtomCategory
        {
            Term = category.Attribute("term")?.Value,
            Scheme = category.Attribute("scheme")?.Value,
            Label = category.Attribute("label")?.Value
        };
    }

    private static DateTime? ParseRFC3339Date(string dateString)
    {
        if (string.IsNullOrWhiteSpace(dateString))
            return null;
```

```
        try
        {
            return DateTime.Parse(dateString,
                null,
                System.Globalization.DateTimeStyles.RoundtripKind);
        }
        catch
        {
            return null;
        }
    }
}
```

1.7 Usage Examples

1.7.1 Example 1: Reading an RSS Podcast Feed

```
using System;
using System.Linq;
using System.Net.Http;
using System.Threading.Tasks;

public class RSSPodcastExample
{
    public static async Task Main()
    {
        using var httpClient = new HttpClient();
        httpClient.DefaultRequestHeaders.Add("User-Agent", "PodcastReader/1.0");

        // Fetch RSS feed
        string feedUrl = "https://example.com/podcast/feed.xml";
        var xmlContent = await httpClient.GetStringAsync(feedUrl);

        // Parse feed
        var feed = RSSFeedParser.ParseRSS(xmlContent);

        // Display feed information
```

```
        Console.WriteLine($"Podcast: {feed.Title}");
        Console.WriteLine($"Description: {feed.Description}");
        Console.WriteLine($"Author: {feed.ItunesAuthor}");
        Console.WriteLine($"Type: {feed.ItunesType}");
        Console.WriteLine($"Explicit: {feed.ItunesExplicit}");
        Console.WriteLine($"WebSub Hub: {feed.WebSubHub ?? "Not available"}");
        Console.WriteLine($"\nEpisodes: {feed.Items.Count}\n");

        // Display episodes
        foreach (var item in feed.Items.Cast<RSSFeedItem>().Take(5))
        {
            Console.WriteLine($"Episode {item.ItunesEpisode}: {item.Title}");
            Console.WriteLine($"  Published: {item.PublicationDate:yyyy-MM-dd}");
            Console.WriteLine($"  Duration: {item.ItunesDuration}");
            Console.WriteLine($"  Summary: {item.ItunesSummary}");

            if (item.Enclosures.Any())
            {
                var enc = item.Enclosures.First();
                Console.WriteLine($"  Media: {enc.Type} ({enc.Length / 1024 / 1024} MB)");
                Console.WriteLine($"  URL: {enc.Url}");
            }

            Console.WriteLine();
        }
    }
}
```

1.7.2 Example 2: Reading an Atom Blog Feed

```
using System;
using System.Linq;
using System.Net.Http;
using System.Threading.Tasks;

public class AtomBlogExample
{
    public static async Task Main()
```

```
    {
        using var httpClient = new HttpClient();
        httpClient.DefaultRequestHeaders.Add("User-Agent", "BlogReader/1.0");

        // Fetch Atom feed
        string feedUrl = "https://example.com/blog/atom.xml";
        var xmlContent = await httpClient.GetStringAsync(feedUrl);

        // Parse feed
        var feed = AtomFeedParser.ParseAtom(xmlContent);

        // Display feed information
        Console.WriteLine($"Blog: {feed.Title}");
        Console.WriteLine($"Subtitle: {feed.Subtitle}");
        Console.WriteLine($"Last Updated: {feed.LastUpdated:yyyy-MM-dd HH:mm}");
        Console.WriteLine($"WebSub Hub: {feed.WebSubHub ?? "Not available"}");
        Console.WriteLine($"\nPosts: {feed.Items.Count}\n");

        // Display blog posts
        foreach (var item in feed.Items.Cast<AtomFeedItem>().Take(5))
        {
            Console.WriteLine($"Title: {item.Title}");
            Console.WriteLine($"Authors: {string.Join(", ", item.Authors.Select(a => a.Name)
            Console.WriteLine($"Published: {item.Published:yyyy-MM-dd}");
            Console.WriteLine($"Updated: {item.LastUpdated:yyyy-MM-dd}");
            Console.WriteLine($"Summary: {item.Summary?.Text}");
            Console.WriteLine($"Link: {item.Link}");
            Console.WriteLine($"Categories: {string.Join(", ", item.Categories.Select(c => c
            Console.WriteLine();
        }
    }
}
```

1.7.3 Example 3: Universal Feed Reader (RSS or Atom)

```
using System;
using System.Net.Http;
using System.Threading.Tasks;
```

```
using System.Xml.Linq;

public class UniversalFeedReader
{
    public static async Task<FeedChannelBase> ReadFeedAsync(string feedUrl)
    {
        using var httpClient = new HttpClient();
        httpClient.DefaultRequestHeaders.Add("User-Agent", "UniversalReader/1.0");

        var xmlContent = await httpClient.GetStringAsync(feedUrl);

        // Detect feed type
        var doc = XDocument.Parse(xmlContent);
        var rootElement = doc.Root.Name.LocalName.ToLower();

        if (rootElement == "rss")
        {
            Console.WriteLine("Detected RSS 2.0 feed");
            return RSSFeedParser.ParseRSS(xmlContent);
        }
        else if (rootElement == "feed")
        {
            Console.WriteLine("Detected Atom feed");
            return AtomFeedParser.ParseAtom(xmlContent);
        }
        else
        {
            throw new InvalidOperationException($"Unknown feed format: {rootElement}");
        }
    }

    public static async Task Main()
    {
        // Example usage
        var feed = await ReadFeedAsync("https://example.com/feed");

        Console.WriteLine($"\nFeed Type: {feed.FeedType}");
        Console.WriteLine($"Title: {feed.Title}");
        Console.WriteLine($"Description: {feed.Description}");
        Console.WriteLine($"Items: {feed.Items.Count}");
        Console.WriteLine($"Last Updated: {feed.LastUpdated}");
    }
```

```
}
```

1.8 Key Features

1.8.1 1. Type Safety

- **Strongly-typed properties** for all feed elements
- **Nullable types** (`DateTime?`, `int?`, `bool?`) for optional fields
- **Enumerations** for fixed-value fields (`FeedType`)
- **Collections** initialized to prevent null reference exceptions

1.8.2 2. Standard Compliance

- **RSS 2.0**: Complete specification implementation
  - All standard channel and item elements
  - Full iTunes podcast namespace support
  - RSSCloud and WebSub discovery
- **Atom (RFC 4287)**: Full IETF specification
  - Person, Link, Category constructs
  - Content and Text constructs with type support
  - Source metadata for aggregated content

1.8.3 3. Extensibility

- **Abstract base classes** (`FeedChannelBase`, `FeedItemBase`) enable custom implementations
- **Easy namespace extension** - add new properties to derived classes
- **Polymorphic design** - work with base classes or specific implementations

1.8.4 4. Date Handling

- **RFC 822** parsing for RSS dates (e.g., `Fri, 10 Oct 2025 12:00:00 GMT`)
- **RFC 3339** parsing for Atom dates (e.g., `2025-10-10T12:00:00Z`)
- **Timezone-aware** parsing with proper UTC conversion
- **Graceful failure** - returns `null` for invalid dates instead of throwing

1.8.5 5. Media Support

- **MediaEnclosure** class for podcasts, videos, and attachments
- **Duration support** via `TimeSpan` for multimedia content
- **File size** tracking with `long` type for large files
- **MIME type** identification for content type detection

1.8.6 6. iTunes Podcast Extensions

- **Complete coverage** of iTunes podcast namespace
- **Episode/season numbering** with nullable integers
- **Explicit content flags** with boolean values
- **Duration parsing** supporting HH:MM:SS or seconds formats
- **Artwork URLs** for podcast and episode images
- **Episode types** (full, trailer, bonus)

1.8.7 7. WebSub Integration

- **Automatic hub discovery** from feed links
- **Self-reference extraction** for subscription topics
- **Ready for push notifications** - hub URLs easily accessible
- **Works with both RSS and Atom** feeds

1.8.8 8. Error Handling

- **Validation** throws meaningful exceptions for invalid feeds
- **Graceful parsing** returns `null` for missing optional elements
- **Try-parse patterns** for numeric and date conversions
- **Informative error messages** for debugging

1.9 References

1.9.1 Specifications

1. **RSS 2.0 Specification** - Harvard Berkman Center
https://cyber.harvard.edu/rss/rss.html

2. **Atom Syndication Format (RFC 4287)** - IETF
   https://tools.ietf.org/html/rfc4287
3. **iTunes Podcast RSS Extensions** - Apple Developer
   https://help.apple.com/itc/podcasts_connect/#/itcb54353390
4. **WebSub Specification** - W3C Recommendation
   https://www.w3.org/TR/websub/

1.9.2 Related Documents

- **02. Analyzing Atom and RSS Specifications** - Feed structure analysis
- **03. Additional Metadata for RSS and Atom Feeds** - Dublin Core, Media RSS, and other extensions

---

Document created: October 25, 2025 | Version: 1.0
Part of the Feed Architectures and Protocols series