

Xtags *MAXIMIZED*

Darryl J. Keck



Xtags Maximized

by Darryl J. Keck



High Volume Press
3500 Dodge St. #200
Dubuque, IA 52003

To report any errors or omissions, email: highvolume@mchsi.com

Copyright © December 2006 by High Volume Press, Dubuque, IA 52003.

ISBN 0-9762041-2-6

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Production and Layout: Darryl J. Keck

Cover and Interior Design: Paula J. Keck

Printing: Midland Press Corporation, Davenport, Iowa

Production Notes

Xtags Maximized was produced using QuarkXPress 6.5, Xtags 6.2, BBEdit 8.5.1, Adobe Photoshop, and Adobe Illustrator. **Computers used:** Power Mac Dual 1.25GHZ G4 and Powerbook Titanium G4. Screenshots were created using SnapzPro X. Photos courtesy of author, www.photos.com, or Art Explosion.

Examples: The examples shown within this book are not part of any other printed book. They were created as similar examples to types of books I've worked with in the past or that I made up for demonstration.

Rights and Liability

All rights reserved. No part of this book may be transmitted or reproduced in any form without the prior written permission of the author. For permission contact highvolume@mchsi.com.

Quark, QuarkXPress and other Quark, Inc. trademarks are the property of Quark, Inc. and all applicable affiliated companies. Portions © 2000 Quark, Inc. All rights reserved.

FileMaker Pro screenshots are provided by permission by FileMaker, Inc. © 1998-2006 Filemaker, Inc. All Rights Reserved. FileMaker is a trademark of FileMaker, Inc., registered in the U.S. and other countries, and the file folder logo is a trademark of FileMaker, Inc.

The information contained in this book is presented without warranty. Although safeguards have been taken while producing this title, neither the author or publisher shall have any liability to any person or company for any loss or impairment caused or suspected by the instructions and examples contained in this book or by the software and XTensions described within.

Trademarks

Trademarked and registered names are used all through this book. Instead of placing a registered or trademark symbol at each incident, we affirm using the names only for the purposes of describing the software or hardware with no intention of copyright infringement.

Xtags is a trademark of Em Software. QuarkXPress is a registered trademark of Quark, Inc. Autopage is a registered trademark of KyTek, Inc. Macintosh is a trademark of Apple Computer, Inc. Photoshop and Illustrator are registered trademarks of Adobe Systems Incorporated. BBEdit, and Microsoft Word are either trademarks or registered trademarks. Other product names or software used in this book may be trademarks or registered trademarks of their respective owners.

Contents

About the Author vii

Acknowledgements ix

Brief Introduction xi

I Introduction to Xtags

Why Use Xtags? 1

Increase Your Profitability 3

Misconceptions 4

Increase Your Value as a Pager 5

Step-by-Step Analysis 6

Create an Effective Coded Sample 7

How to Use This Book 8

Written for QuarkXPress 9

Knowledge of Scripting 9

Final Note Before Starting 10

2 Getting to Know Xtags

Setting the Preferences 11

Don't Export Style Sheet Definitions 11

Omit Default Elements in Xtags List Tags 12

Output separate tags (e.g., <I> vs. <BI>) 13

Delimiters 13

Get Text and Save Text with Xtags 13

Get Text with Xtags 13

Importing with Autopage 15

iv *Xtags Maximized*

<i>Save Text with Xtags</i>	15
<i>Exporting with Autopage</i>	15
<i>AppleScript Get and Save Text</i>	15
Copy and Paste Xtags	16
<i>Copy Xtags Text</i>	16
<i>Paste Xtags Text</i>	17
Style Sheet Tag Attributes	18
<i>Character Information in the Style Sheet Definition</i>	24
<i>Creating Character Styles</i>	26
<i>Style Sheet Overrides</i>	26
Paragraph Style Sheet Examples	27
Xtags Export Issue	31
Changing the Inset (for Autopage 5.8)	32
French Quotes vs. Brackets	32
Translation Tables	32
<i>Translation Table Start Tag</i>	33
<i>Start with a Template</i>	34
<i>Translation Table Codes</i>	35
<i>Translation Table Format</i>	35
<i>Translation Table Issue</i>	36
<i>Test the Codes Before the Translation Table</i>	36
Master Pages	37
Troubleshooting Tips	38

3 **Building Picture and Text Boxes**

Xtags Fields	41
Xtags Issues	42
Unanchored Picture Box Tags	43
Unanchored Text Box Tags	50
Xtags Caption Positioning	51
<i>Side Caption (Top Left)</i>	52
<i>Side Caption (Top Right)</i>	52
<i>Top Caption</i>	53
<i>Bottom Caption</i>	54
<i>Side Caption (Bottom Left)</i>	54
<i>Side Caption (Bottom Right)</i>	55
<i>Grouping</i>	56
Anchored Picture and Text Boxes	56
<i>Anchored Picture Boxes</i>	56
<i>Anchored Text Boxes</i>	57

<i>Anchored Picture Box Examples</i>	57
<i>Anchored Text Examples</i>	58
Don't Exceed the Page Size of Document	61
Importing on the Pasteboard	61
Unanchored Lines	61
<i>Unanchored Line Example</i>	64
Anchored Lines	65
<i>Anchored Line Example 1</i>	65

4 Image and Caption Building

Shrink to Fit Capabilities	67
Relative Caption Placement	68
<i>Indented Relative Placement</i>	69
<i>Vertical Relative Placement</i>	70
<i>Vertical and Horizontal Mixed Relative Placement</i>	70
<i>Fit-to-Height, Fit-to-Width</i>	73
<i>Forced Anchored Leading</i>	75
<i>Additional Height Options</i>	76
Working with Figures	76
<i>Space Between Rule and Figure</i>	76
<i>Rule Around Picture and Caption</i>	77
<i>Labels Below Art (Alphas)</i>	79
<i>Bringing in Multi-Piece Figures Together</i>	80
Complicated Figure Caption Usage	82
<i>Side Element with 5 Grouped Boxes</i>	82
<i>Relative Double Border</i>	84
<i>Caption Top, Source Line Bottom</i>	85
<i>Working with Blends in Captions</i>	86
<i>Adding a Number on the Figure</i>	88
<i>Rotated Solid Shadow on Image</i>	89
<i>Shadow on Images Shrinking-to-Fit Both Dimensions</i>	90
<i>Screened Caption Under Image</i>	93
<i>Screened Caption Below Image</i>	94
<i>Fit-to-Height and Fit-to-Width Multi-Piece Example</i>	95
<i>In Closing</i>	96

5 Creating Boxes and Elements

Origin Tags	97
<i>Pre-paging with Origin Tags</i>	99

vi Xtags Maximized

Creating Boxes with Xtags	101
Single-Column Box	101
Single-Column Box with Rotation	103
Simple Box with Background Art	104
Example of a Two Box Approach with Relative	105
Rounded Corner Box with Unsupported Shape	106
More Advanced Unsupported Shape Box	107
Combining Elements	109
Side Margin Box with a Vertical Line	110
Working with Multi-Column Text Boxes	111
Inline Text Elements	113
Table with Art Handling	114
Complicated Box with Art	115
Complex Box with Seven Xtags Strings	117
Complicated Box with Rotated Text	119
Side Margin Box Expandable	121
Picture Box Using Skew	123
Using Lines for Top and Bottom of Box	124
MathType Anchored with Xtags	125
Oval Text Box with Offset Shadow	127
Complicated Box with Multiple Elements	128
Expand or Shrink-to-Fit with Graduated Screen	130
Box with 3 Separate Parts	131
Unique Box Treatment Using Skew and Angle	133
Using Xtags for Design Element in Box	134
Using all the Placements in One Element	135

6 Custom Publishing

Some Background on Grep Searches	138
Looking at the Operators	139
Custom Publishing Using Xtags	141
The Opening Page	142
Adding Autopage Codes with “Grep”	152
Altering Boxes to New Style	154
In Closing	156

7 Xtags Working with Autopage

The Paginate Window in Autopage	158
Repurposing Content	159

Side Art – Using Reference	160
Rotated Tables	162
Left and Right Art Placement Option	163
<i>Solution 1: Using Xtags with Left/Right Art Placement</i>	164
<i>Solution 2: Manually Positioning the Rule</i>	166
Text Wraps (Runarounds in Autopage)	168
<i>Method 1: Treating Wrapping Art as Side Art</i>	168
<i>Method 2: Using an Anchored Box</i>	170
Oversized Two-Column Inline Art	172
<i>Handling After Import</i>	172
<i>Bringing in with Xtags</i>	172
<i>Anchoring the H1 Side Head Instead of Side Art</i>	174

8 Repurposing with Xtags

More Intricate Coding	183
Working with Heads and Run-in Heads	186
Images	186
Searching Through the File	187
Closing the File	188
Know the Content	188
Floating Element Markers	189
Hyperlinks	192

9 Xtags and FileMaker® Pro

Level of Understanding	194
Importance of Databases	194
Database 1 — Xtags Creator	195
<i>Scripting Option</i>	197
Database 2 — Style Sheet Generator	210
Database Production Benefits	216

<i>Xtags 7.3 and Beyond</i>	217
-----------------------------	-----

<i>Index</i>	225
--------------	-----

Programs of Choice:

When I read anyone's software book, I always like to know the author's favorite programs to work with. Here's my list:

Paging:	QuarkXPress w/Autopage XT
Text Editor:	BBEdit 8.5
Database:	FileMaker Pro
Art Programs:	Adobe Photoshop CS2 and Adobe Illustrator CS2
Screen Captures:	Snapz Pro X
Web Creation:	BBEdit 8.5, Dreamweaver MX
Programming:	AppleScript, MacPerl
XTensions:	Xtags, Autopage, and Headers & Contents

Ordering Information:

To order any of the software covered in this book, go to:

Xtags:	www.emsoftware.com
Autopage:	www.kytek.com
XMLxt:	www.kytek.com
QuarkXPress 7:	www.quark.com
FileMaker® Pro:	www.filemaker.com
BBEdit 8.5:	www.barebonessoftware.com

Special Thanks to:

Chris Ryland and Em Software, Chris Roueche, Keith Erf and KyTek, Inc., Benjamin Ko, Joe Fuentes and Quark, Inc., Ramona Percelle and FileMaker, Inc., Sara Niki Thomas, Patrick Woolsey and Bare Bones Software, John Ferguson and Kinetic Publishing Services, Ronni Burnett and McGraw-Hill, Pearson Education, Paula, Brandon, and Chandler Keck

About the Author

I have been in the publishing industry for nearly 20 years. My first experience with Macintosh was working with the *Tucson Entertainment Magazine* in the evenings. I quickly learned a few things about the Macintosh and how it was a time saver compared to traditional paste up. Within a year, I started my own internationally distributed music magazine, which I decided early on would be driven by the Macintosh. Although the software was quite primitive, I knew this computer system would quickly develop and I would grow with it. I began my magazine using PageMaker, and quickly moved to QuarkXPress upon the introduction of the 2.12 version.

I spent time producing music publications until the early '90s and then moved onto collectible magazines and designing collectible books for *Antique Trader Publications*. *Antique Trader Publications* was a great place where I did all the design and paging on the book, including art, using Adobe Photoshop and Aldus Freehand. No employer I've since worked with allowed that much creativity and versatility. The need for learning so much software made it very possible to pick up other programs quickly.

Following *Antique Trader*, I took a position at *Times Mirror Higher Education (TMHE)* in its pagination department. At *Times Mirror/Wm C. Brown Publishing*, I again worked with QuarkXPress, but I also saw the use of Penta and thought that Quark really needed something code-driven that would automate some of the manual work we were doing.

After leaving *TMHE* to return to *Antique Trader Books*, I ran into a former colleague from *TMHE* who told me of a new position she held at a local composition company. I began working there shortly after where was introduced to Xtags. In my spare time, I would push the program as far as I could take it. I'd search through previously paged books looking for boxes, tables, or anything that looked challenging and figure out a way to achieve them, even if it required additional scripting.

I've always had a passion for writing and in late 2005, I wrote my first book on the program *Autopage* titled, *Autopage 6: Automating the Pagination Process*.

x *Xtags Maximized*

This book has received a lot of positive feedback from pagers and setup specialists all over the world. Upon writing the first book, I knew that I would one day write a book on Xtags because it has always been one of my favorite software packages to use. I have written two other books in the past year and this being my second software title. I also write screenplays, when time permits, with one titled *Dream Theater*, which was released as a book this past August.

I currently run my own company, *High Volume Press*, where I concentrate on self-publishing, as well as producing textbooks through paging and layout, Autopage and Xtags setup, scripting, design, custom publishing, consulting, training, repurposing, and database creation.

Darryl J. Keck

Introduction

I began writing *Xtags Maximized* almost as soon as I completed the *Autopage 6: Automating the Pagination Process* book back at the beginning of 2005. Chris Ryland from Em Software had discussed with me the possibility of writing an Xtags book and I liked the idea of it, but I had just started working as an automation specialist for a company which I knew was going to take most of my time. Therefore, it was put on the back burner for quite awhile. Whenever I'd think of something new or unique concerning Xtags, I would write it down in a notebook or enter it on my laptop. Before long, I had so much information that I couldn't keep myself from getting started.

My schedule cleared up enough where I could really start giving this title my full attention. The more I worked on it, the quicker I realized that this was going to be such a rewarding book to write. Not only for myself, but for the Xtags user that really wants to push the envelope and not just use it at the basic level. Every company always has that person that sees an XTension like this and wants to take it to the next level. This book should help you reach that potential.

Even though I'm at an advanced level with Xtags, it's still a challenge to write this text because getting an application to come across as interesting is very important to keep the reader motivated. Xtags is one of the most powerful QuarkXPress XTensions I've come across, yet it's one that a lot of people get intimidated by. I feel those who run into problems generally aren't visualizing where they want to go with it. My goal here is to break it down simply, emphasizing all of the advantages to using this over manually creating boxes, art, and other aspects that this is designed for. I would also like to see those intermediate-level users acquire an advanced skill level by acquainting them with new workarounds they weren't previously aware of.

Xtags is a very affordable XTension that can easily pay for itself just on the art importing capabilities alone. On most titles, my time would improve anywhere from 30% to 60% over manual paging by using Xtags as well as Au-

topage. The more ways you find to automate, the higher the profits you can potentially generate. By investing in Headers & Contents, the overall process gets even more streamlined. Most people will feel that statement isn't correct, but using many XTensions to automate previously manual tasks ensures more profitability. I guarantee that the prepress dollar will continue to shrink as work is farmed out of the country.

One of my goals for this book is to expand on the uses of Xtags and to take away the apprehension some feel when using the program. This book does not replace the *Xtags User Guide* that Em Software provides when purchasing the program. Although I obviously cover aspects of the program that reside in the *Xtags User Guide* by Em Software, I feel that my presentation takes a different approach to the same material. Where this book differs, in my opinion, is that it not only contains a multitude of examples, screen captures, and so on, but also contains information from the hands on set up, paging, or repurposing of approximately 450 books—I have run into almost every scenario imaginable. I'm trying to inform the reader in a way that a pager or setup person will understand. If nothing else, I want to take the guesswork out of Xtags and help others get the results I do with this great software as well as some unique workarounds for tasks normally handled manually.

I want to show users how Xtags will totally change their workflow and give them an edge. It's very important to find ways to make the paging process more efficient. Xtags is an important solution to that problem. Using Xtags to import art into your file is one of the biggest time savers I've encountered. What normally would take a person a half hour can be reduced to about 30 seconds with the program's shrink-to-fit and relative placement capabilities.

While this book is very comprehensive, it does not cover every aspect of the program. Several things will only briefly be touched on, but for more detailed descriptions, the *Xtags User Guide* will be the best place for that information. This book is geared more for those who use the program on a daily basis and want to find effective ways to add functionality to their paging workflow. I feel that between this book and the *Xtags User Guide*, all the bases should be covered.

I'm not claiming that every Xtags scenario is in this book. Ideally, I would like to have caught everything, but chances are good that I didn't because the combinations are so vast when designing. I do feel like I've put forth a multitude of step-by-step examples that should improve your knowledge of Xtags and offer new possibilities you may not have thought of. If I get feedback from a few pagers telling me this book has helped them, it will be worth all the effort that went into writing it and creating the examples to learn from.

1

Introduction to Xtags

CHAPTER

I remember hearing the phrase...“A fool and his money will soon part.” I feel that the phrase “Only a fool will bring in art manually when Xtags™ exists” should be coined. Xtags 7.3 for Quark 7, Xtags 6.2 for Quark 6.5, and Xtags 4.2 for Quark 4.1.1, is a text and image importing/exporting XTension by Em Software that has a much broader scope than XPress Tags in QuarkXPress. With Xtags, you can write code to import floating and anchored art, create boxes, shrink-to-fit floating and anchored art, override paragraph settings, create a translation table, and much more.

Xtags is a very in-depth program and I’m really hoping that this book shows you how vast the possibilities are. My goal here is to show you some of the best methods for increased productivity. I will say that you will need Xtags if you are using Autopage and want to maximize your productivity. I personally cannot do without Xtags, nor would I.

Why Use Xtags?

Operating QuarkXPress without Xtags is like buying a car without the tires. There is so much importing power in this XTension that not using it would only amount to decreased profits for you or your company. There are a lot of companies out there that own this software that are not using it even close to its potential. They look at a floating element that has difficult elements and decide to put it in the library instead. Then they populate it manually each time it is called out in the text. I’ve witnessed this from others first hand and I always said, “You need to use Xtags more. It will save you hours.” I then would set up some boxes for them and they saw the power first hand.

I will stand by this statement: Bringing in your art and captions manually is throwing money away! To bring in 100 pieces of art manually and align them with the caption could take anywhere from 30 minutes to an hour. One

2 Xtags Maximized

of the important features of Xtags is its ability to import the art grouped with the caption with shrink-to-fit capabilities. To import 100 pieces of art will take the program less than two minutes.

Although you can make detailed macros for character styles, I usually use the QuarkXPress character styles whenever possible without using Xtags for this unless I discover some overrides or if it is necessary to have several character style sheets within one style sheet.

My primary uses for Xtags are the following:

- Bringing in floating art
- Bringing in anchored items (icons, math, etc.)
- Importing complicated tables, boxes, and side art
- Coding Autopage tags instead of markup
- Maintaining my translation table
- Writing advanced macros

Pagers using Xtags to change bullets, heads, and so on could use Quark's character styles for this instead, but it all depends on your workflow. I would use Xtags if overriding a style sheet. For users of XMLxt with a translator, this is a standard practice, although there are times when I need to break the rules as well.

When I wrote my first software book, *Autopage 6: Automating the Pagination Process*, I found one of my favorite aspects of writing it was the section on Xtags. I had allotted about 25 pages for this and quickly realized that I was only scratching the surface on what the XTension offers. Considering how much Xtags has helped me streamline and speed up my workflow, writing a full book was something I looked forward to.

I was talking to a former co-worker when I first thought about writing this book and their response was that they only use Xtags for importing the art using the translation table I created for them. It was at that moment I realized how this software's potential isn't being utilized. Many pagers and set-up specialists don't approach a problem seeking the best solution. They just move on to the easiest resolution which generally is additional manual time spent throughout the project. I personally think that is a counter-productive way to approach any workflow. It's to your advantage to go through this book and learn all that is possible using the code combinations Xtags has programmed within.

I feel that anytime you are creating a book, any floating elements (art, boxes, side art, tables, etc.) that can be automated during import will save production time during the paging process. I am a firm believer that the goal is to save as much time as possible. This means you can handle more projects and make yourself or your company more profits. At this critical point in the paging industry, it's for your best interest to put forth all the effort possible to keep jobs at your company rather than being outsourced to foreign compa-

nies that have economic advantages over you. They may be able to do the work at a fraction of the cost, but if you are using all of the efficiencies at your disposal, you can maintain the work in your company. I feel that the failure to maximize software like this may have a lot to do with the global shift in prepress services going to other countries. You can do something about changing this...here is an opportunity right in front of you.

Don't fall into that paging trap that you cannot find the time to set floating elements initially. The time you set up the project using Xtags will easily be caught up. This usually happens within the first two chapters. The secret is to always use as few text boxes as possible. This is a trick I will teach you throughout this book. Another thing to remember is that Quark is a "what you see is what you get" software, but underneath are codes making this all happen. These codes can all be enhanced using the power of Xtags, AppleScript, and other programs.

Increase Your Profitability

When a person looks at the limited amount of menus that Xtags has within the Quark program, it appears that it might be somewhat limited, but it's quite the opposite. So much can be accomplished in this program that it's quite astonishing. When I look back many years ago before I discovered the power of Xtags, I often wonder how the projects were profitable. The mindset you have to adapt is to be as profitable as possible.

Having worked with Xtags for over 9 years, I have used the codes on hundreds of projects and have proven to be one of the most profitable automation specialists for any company I've worked for merely due to my ability and knowledge of streamlining the process. If you manage a composition department, this program should be mandatory for both InDesign and QuarkXPress. The tools exist within this program to generate increased profits for your company. But just having the program loaded in your XTensions folder won't do anything for you. It's up to you to have your staff read this book, look for efficiencies and take the time to make them happen.

I've always been bothered by the small percentage of paggers in a composition department that actually know the software to an advanced level. I worked in a composition department that had 15 to 20 licenses of Xtags. Only four of us really used it productively. Others would make foolish claims that it was less complicated to manually import the art. I decided to do a test to see how long it would take to import 50 pieces by hand with the captions and 50 using the Xtags (including figuring out the codes and building the translation table). Here are the results:

Manually	18:30
With Xtags (full setup)	4:24
Xtags (import only)	0:39

4 Xtags Maximized

As you can clearly see, the manual import took over 18 minutes and that was including duplicating the picture and text boxes and not even making sure all of the images were snapped appropriately to the x and y coordinates of the box. Xtags not only snaps to the x and y coordinates, but it shrinks to fit the images as well. Think about a book with 1,000 images. This would be close to 8 full hours done manually. To import 1,000 images (including the translation table setup) would take around 20 minutes. This proves how much more productive this process can be once you become effective at this. Add that up over a year and see how much money has been thrown out the window.

Although there is quite a learning curve initially for Xtags, I feel I will make the functions I do on a daily basis more understandable. There is always more than one way to do something. I'm not here to tell you to totally change your current workflow, but you may find that you will gain a higher level of productivity using the many features of Xtags. Once you successfully import art with Xtags, chances are you will never want to manually bring in an image again. I speak from experience on this.

It is important to apply your knowledge of importing art to building boxes, side art, and other floating elements. There are so many efficiencies that can increase the profitability in your company using this program.

Misconceptions

Since using Xtags, I have noticed there are many misconceptions about its uses. It can be used in many areas of QuarkXPress, but does not necessarily do everything and eliminate all manual work. However, it reduces time spent on so many manual tasks such as importing art and creating style sheets that normally would take away from profitability.

Some of the biggest arguments that pagers always use to avoid using the program are shown in this section. I must stress that most of these statements have come directly from pagers who haven't spent enough time learning the program and continue doing manual tasks that should be automated.

"There is just not enough time to set up a book with Xtags. The schedule is too tight. I can't get a coded sample set up with all these codes."

This could be seen as a justifiable excuse to some degree, especially when it is the heaviest time of the production season and you're in the middle of several books. It is sometimes difficult to put aside the necessary time to focus and complete the upfront tasks. However, if the book isn't set up with Xtags, a massive amount of wasted time will be spent manually creating boxes, art, and overriding style sheets. The up-front time spent on coding the sample would have been saved throughout the project. In my opinion, a successful, cost-effective project relies heavily on the up-front setup of Xtags.

“There isn’t that much art in the book. It will be faster to just bring it in manually over creating the translation table.”

I have to admit that there have been times where I thought that would be quicker, but I found that it is still faster to have the art coded for Xtags and then have it import in. It is so much easier to have it auto-import rather than having to open up the server, find the folder, and then import in the images. Even if there are only 5 figures in the chapter, the translation table still is faster. The secret is to have a standard translation table where you just fix the width, depth, and path to the art. I will cover this in greater detail in Chapter 2.

Increase Your Value as a Pager

At a minimum in the upcoming years, a pager will need to be very proficient at the following programs/XTensions:

1. QuarkXPress™ 6.5 and 7
2. Xtags™
3. AppleScript
4. Autopage®
5. FileMaker® Pro
6. XMLxt™
7. BBEdit™ with its text editing capabilities, or MacPerl for having more control over the functionality

If you cannot become skillful with these programs, your value as a pager greatly diminishes. That is why it’s so important to be as competent as possible with these programs. I’ve never bought into the excuse that a person’s avoidance of a program is due to lack of training. Whatever happened to experimentation or using a program’s tutorial (if one is provided)? It is in your best interest to experiment with these programs when time is available. A pager must always be evolving and that motivation strengthens your value as an employee. For a self-employed freelancer, it is invaluable to find ways to keep the profit margin up. There are so many tasks that are overhead that it’s necessary to your survival to find programs that reduce your production time. A little ingenuity can keep you in business much longer.

Furthermore, do not be afraid to tap into the resources around you. Depending on the size of your company, there are inevitably co-workers who excel at these programs and XTensions. Employees who avoid mentoring those advanced users are only cheating themselves. There is usually someone who knows the answer to the question you have. If you cannot find a resolution, there is always the manual or help menu with the program. If you still

6 Xtags Maximized

cannot find your answer, there are plenty of web resources to take advantage of such as “www.google.com.” Go to www.google.com and within *Groups*, type in “Xtags art”. It’s possible there is someone out there who has posted something on this subject.

At the very least, try a few different avenues before giving up on the matter. Troubleshooting will only strengthen your skills and will help you attack each new problem as it arises with greater clarity in future projects. I truly believe that getting any answer you want relies on first asking the question.

Step-by-Step Analysis

I follow a dozen steps prior to paging my first chapter. The document preparation is critical to the success of the book. You can alleviate many potential problems by just knowing where to begin with your document. I always start by having a printed copy of the sample/design. I review this page-by-page, finding places where Xtags and Autopage will need to be used. I will even adjust the file that the designer set up to ensure that boxes, tables, and anchored elements are working the most efficiently.

When setting up a project, it’s extremely important to know where you’re going and not to get dispirited if you can’t get there immediately. Some of my most taxing moments were solved when the computer was turned off. Sometimes this occurred during supper, when I was taking a drive, or I’d wake up with a solution. I’m not trying to imply that your life will turn into one long quest for a resolution, but if you are at a complete loss, sometimes walking away will be your best course of action. It’s amazing how your subconscious will search for the answer you seek. The point is that everything isn’t always clear cut, so don’t get discouraged.

I’ve developed the following method when beginning a book. For a quick rundown, the steps are:

1. Be certain the design’s settings and preferences are to the proper company specifications.
2. Review a copy of the design. Go through each page and decide which elements will require Xtags and Autopage (if you use the program).
3. Mark up each page with the necessary macros, Autopage tags, and Xtags markup. If you do not have a coding department, you will need to use a program like BBEdit combined with AppleScript to get these into your input document.
4. Review the *Quality Control* printout from Autopage.
5. Have a checklist with the variable space bands (VSBs) to fill out.
6. Add the necessary H&Js to the Quark template. Always start with a template of your design. This will reduce any possible errors in consistency.
7. Test the difficult Autopage and Xtags setup to verify your calculations and make sure boxes, figures, and other elements are working correctly.

8. Input the *Short Line Elimination* amounts to the template.
9. Add the Headers & Contents information to the Running Heads. If you do not own Headers & Contents, you will have to skip this step.
10. Build the Xtags strings and create the translation table.
11. Bring in the text for the first chapter and test it to make sure everything works as intended.
12. Build a coded sample showing the translation table codes and how they should be coded.

Create an Effective Coded Sample

Creating an effective coded sample is a necessary early step into the profitability of a project. Depending on your markup staff, it is important that they understand your tagging structure and the proper positioning of your elements. If you have the ability to write a script to put in macros, then a coded sample may only be necessary for yourself for use as a checklist.

If scripting isn't a possibility, putting forth detail will help the coding department decipher your intentions. It's important for the coded sample to be using two sets of instructions: one box showing style sheets for codes on the side, and one for listing the macros, Autopage tags, and Xtags coding instructions. Here is an example of how the coded sample should look:

46 Part 1 • Coral Reefs

T

Black sand beaches are very common in parts of Hawaii. There's Moana Loa and Moana Kea and there's the Road to Hana which is part of the Maui landscape, but a treacherous road takes you to it. Not for the faint hearted, but worth the drive if you have the time and nerve.

H1

[[SR H1 V=2 L=H1_BAR]]

Pacific Coral Reefs

[[ES]]

H2

South Pacific Coral Reef

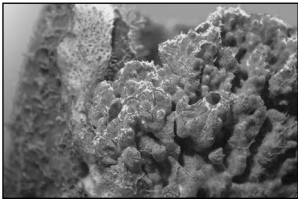
T1

Reefs in the South Pacific are rarely touched due to the lack of tourism outside of the Hawaii region. Black sand beaches are very common in Hawaii. There's Moana Loa and Moana Kea and there's the Road to Hana which is part of the Maui landscape, but a treacherous road will take you to it. Not for the faint hearted, but worth the drive. The scenery along the way is nothing short of breathtaking.
The most common type of lava-seawater explosion is known as Tephra Jets. There's also Punahū Black Sand Beach in Hawaii.

★★ Use Macro [[p1]]Art Name[[p2]]caption[[p3]] for Photos

Some coral reefs in the South Pacific are believed to have survived for millions of years

PAC



Pacific Coral Reef 47

Makalawena Beach contains a rocky area and the sandy beaches features a breathtaking views of mountains in the distance.

H3

Hazards to Coral Reefs

<@Sp></></>

Coral reefs have been known for years as support for all marine life, yet are being destroyed due to pollution, erosion, and other factors, most caused by humans. Many are out to protect coral reefs by using enforcement policies. Most of these are maintained by fisherman and biologists who have an eye out for dangers to them.

H4

Ecosystem of Coral Reefs

★★ Use Char Style <@H4>

Coral reefs are one of the most intricate ecosystems in nature. Some are believed to have survived for millions of years, but due to pollution, erosion, and other factors, most caused by humans. Many are out to protect coral reefs by using enforcement policies. The reefs are important to plants, fish, and any life form dependent on them.

★★ Use [[TR B1]] at reference.
Type [[b1]][[T B1]] at start of box before [BT] style and [[b2]] at the end of the box.

BT

Rock Mass Under the Sea

B1

Rock mass under the sea is caused many different factors. Cliff leads to Papakolea Green Sand Beach. For very private times, Pine Trees Beach is a favorite for travelers who are looking for tranquility over mass appeal. Spencer Beach Park is protected by a giant reef.

BEX

In Maine, the ocean temperature rarely exceeds 65 degrees in the hottest parts of the summer.

B

Sand Beach is nestled between mountains and rocky shores. For tranquility over mass appeal. Spencer Beach Park is protected by a giant reef. Makalawena Beach contains a rocky area and the sandy beaches features one of the most incredible views of mountains in the distance.

The coded sample should have a mixture of character styles, Xtags, and Autopage codes. I incorporate as many strings into Xtags as possible. My translation table can end up being very involved, but it keeps markup errors

8 Xtags Maximized

to a minimum. I also put my AutoTags into my translation table. This way the coder would only see a marking like `[[LC]]` in the coded sample, but my translation table would have

```
[[LC]]      [[LC 1 M=36p A=4 S=.5 H="Cyan"]]  First Layout Change
```

One of the secrets of the Xtags experience is that a lot of power should center around two places: the translation table and the ability to get the tags in the coded file. It is not cost-effective to have the coder put in the long codes, so make sure you always use compact markers. For figures, it is best to keep them to something short like:

```
[[fg1]]Figure 1.2[[fg2]]
```

This makes it easier for the coder and saves the company money. Having this broken down will make the project easier for those handling it. In the Xtags section, I will cover in more detail how to make your coded sample easier using the power of the translation table.

How to Use This Book

One of the prerequisites to be successful with this title is that you have used Xtags on at least one project. Also, I feel it is necessary that you have read through the *Xtags User Guide* by Em Software, which is an invaluable reference guide. It's obvious to me that there would not be a need for this book if I did not have a distinguishing approach to the material. Where I feel this book is unique is through my step-by-step approach, helping you achieve the task without having to keep referring back to certain sections to pull it off. I always have learned through instruction like this and I more often retain something once I do it. It locks in the memory better than just reading about descriptions of various tasks. I want to help people who sometimes get a little confused on where to begin. The more you have working correctly up front, the less chance there will be of unforeseen problems later in the project.

This book has not covered every aspect of the program because that is not my intention. Some facets of the program like “macros” I will not be touching on at all because that is not part of my workflow. I use scripts to do these functions instead which I will be showing you throughout this title. I also feel that the *Xtags User Guide* does an incredible job with their coverage of macros.

My main objective is to show the user how to get more out of the program by approaching Xtags in a different manner than the user might be used to. I will also show how to increase your productivity by incorporating BBEdit and AppleScripts and into your workflow. I feel if intermediate and advanced users work through the examples, they will find themselves more proficient than before. Many of these concepts you may be familiar with, but my hope is that you will learn many new and important ideas along the way.

Written for QuarkXPress

This book was written with QuarkXPress 6.5 and 4.1.1 in mind. I realize that Xtags has a version available for Adobe InDesign®, but I have not used the InDesign version enough, at this point, to really be able to set up examples to help users become more efficient. Depending on the feedback I receive, I may write an InDesign specific Xtags book in the upcoming months, but for the time being this is primarily focusing on the QuarkXPress 6.5 version. I have noticed in testing that most of the picture box and text box strings seem to work between both programs, but since my workflow has been primarily based around QuarkXPress, I have chosen to focus primarily on that platform.

I do feel this book will help users of both programs, but you will have to know where the limitations are concerning both versions. The *Xtags User Guide* does have specific “InDesign” differences and is where you will need to locate what the unique “InDesign Only” functions are. Pay special attention to the **InDesign Caveats** throughout the *Xtags User Guide*. The **InDesign Caveat** illustrate specific differences between the QuarkXPress and InDesign versions.

Knowledge of Scripting

Throughout this text, there are quite a few references regarding AppleScript and occasionally Perl. If you do not understand these scripting languages, it doesn't mean that you won't get enough out of this book because you will. I do feel if you can understand these, you will be getting even more because it will open the door to new possibilities and increased automation.

AppleScript is easy to understand and with programs like BBEdit that have so many recordable functions, it is very user-friendly. This book does not break down AppleScript and teach it, but it will show you how to improve your workflow using it.

If you have a task needing accomplished, the easiest way to learn this is reviewing someone else's AppleScript. It is helpful to the learning process by how many sample scripts are posted online for you to borrow and use. You can usually find what you are looking for or something close enough. Once you have it, you can amend it to fit your needs.

This book will show many AppleScripts and how to increase automation by using these examples. Many of these functions are through BBEdit, while others are specific to QuarkXPress. My purpose for including them is that they are part of my current workflow. I feel that it is important to script as much as possible. Do not be afraid if what you have written is not the most proficient method. I am frequently finding more effective methods, but that is part of the growing process in any workflow. Just do what you can to achieve the results. You can always streamline your code later when your understanding develops.

Final Note Before Starting

Over the years I have been employed at different companies and I've observed many people excel at different programs, but you should not get discouraged if you do not gain an understanding as quickly as someone else. Not everyone who sits down with a guitar becomes a virtuoso, but I think if you follow along step-by-step and experiment with the examples I am showing you, you will be above where you expected to be.

This book can really give you an edge if you actually read it and do the work. My belief is that software books collect dust on the shelves of composition companies everywhere. It is always the 15% of the workforce that will pick up a book and actually read it. Those same people are generally the ones who end up with better raises, new opportunities, and the best projects.

Don't let others tell you it cannot be accomplished if you feel otherwise. I've been the victim of that too many times. I have worked in a production department where I constantly heard people say, "This cannot be done. It's a manual thing." When I was new to the department, I would sit back and say to myself that it can be done. I made the mistake of not challenging them and that was probably a bad idea in hindsight. It is always best to let others know what you have discovered because it will quickly become part of the workflow if others realize it can be accomplished.

It is a good idea not to be intimidated by the tags. When first looking at an Xtags string, it can seem overwhelming, but you need to look at each comma as a field. If you've ever worked in a database program, try to picture each field like that and it may become easier to grow more comfortable with. Regardless of your past experience with Xtags, it's time to challenge yourself and gain a new understanding of what is possible.

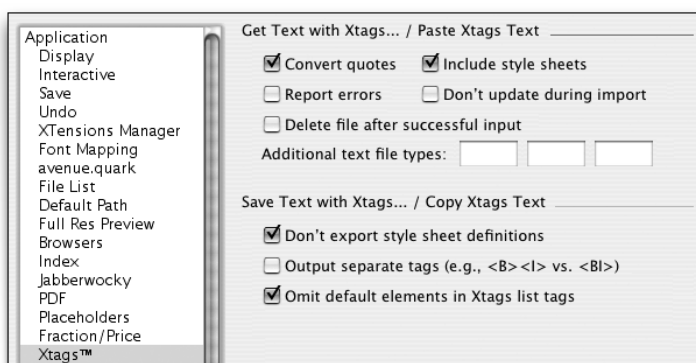
2 Getting to Know Xtags

CHAPTER

Xtags has many of the same features as XPress Tags in QuarkXPress, but can handle so much more. A lot of power resides in the style sheet tags and we need to really dive into this. Most of the text editing features will be covered in this chapter. We'll get more into the text and picture box tags in the next chapter. What I'm not trying to do here is to rewrite the *Xtags User Guide*. Some aspects of this program I will touch on only slightly, whereas others I will go into more detail. Let's start with the Preference options and ways to import your Xtags.

Setting the Preferences

It is very important to get the Xtags Preferences set up to the way that works best for you. There are many selections to choose from. Currently, I set up my Xtags Preferences as shown here:



Don't Export Style Sheet Definitions

It's clear to me that everyone uses the program differently. If I have my style sheets set up ahead of time, I wouldn't want the style sheet information

12 Xtags Maximized

to export when I do a *Copy Xtags Text, Paste* combination. I make sure I check the *Don't export style sheet definitions* in the preferences.

When this is checked, during the *Copy Xtags Text*, the code appears as:

```
<&o(0,0)><&pbu2(0,0,198,39,0,0,,n,0,(K,n),(100,100),"Solid",n,
100,0,m,100,100,0,0,0,0,"56 GB Disk:AP_6.0:AP_Book:
Art:ch03_009.eps" , "" , "" )>
```

If this is unchecked, the code will contain extra style sheet information that isn't pertinent to the tag unless you have some local style information that you want to use or if the style is undefined.

Before putting this in the translation table, you will have to delete the unneeded information, so it's a good practice to leave this checked unless you have a system that formats your style sheets. I will talk more about this later in the chapter. Here is the same Xtags information with *Don't export style sheet definitions* unchecked:

```
@Normal=<Ps100t0h100z12k0b0cKf"Helvetica">
@DT=[S"" , "DT"]<*J*h"Standard"*kn0*kt(2,2)*ra0*rb0*d0*p(24
,0,24,12.5,12,12,g,"U.S. English") Ps100t0h100z10.5k0b0cKf"
Berkeley-Medium">
<&o(0,0)><&pbu2(0,0,198,39,0,0,,n,0,(K,n),(100,100),"Solid",n,
100,0,m,100,100,0,0,0,0,"56 GB Disk:AP_6.0:Xtags:Art:
ch03_009.eps" , "" , "" )>
```

The style sheet is bringing in the “DT” tag and all of its attributes. This can be very necessary information, but if predefined, it is not necessary to include all of this. You need to know when you should leave this checked.

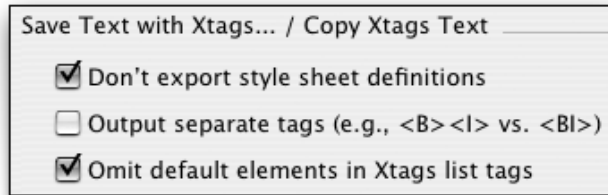
I'm not suggesting that you need to select *Don't export style sheet definitions*. I find that in a lot of cases, I do like to see what the style sheet information is. I will sometimes run an AppleScript against it to flag styles that might have errors in them. For example, I always like to have my *Keep Lines Together* (2 and 2). So I may run a script that runs through and generates a list that shows any style that has <*kt0> or <*ktA> applied. This can also work if you want to see if any stray fonts are showing up in your styles by running a script against the font part of the tag.

Omit Default Elements in Xtags List Tags

This option is also important depending on your comprehension level of the program. When pasting the text following the *Copy Xtags Text* feature, you will normally have this string:

```
<&pbu2(0,0,198,39,0,0,,n,0,(K,n),(100,100),"Solid",n,100,0,m,1
00,100,0,0,0,0,"56 GB Disk:Xtags:Art:ch03_009.eps" , "" , "" )>
```

Some people do not like all of this extra information in the path because the majority of the fields are default information. These can be excluded by selecting the *Omit default elements in Xtags list tags* in the preferences:



When doing a *Copy Xtags Text, Paste* combination, this is the result:

```
<&pbu2(0,0,198,39,,,n,(,n),(,100),,n,,,m,,,,,"56 GB Disk:Xtags:
Art:ch03_009.eps",,,)>
```

The path is much cleaner, but I must caution that it's more difficult to find the needed fields without counting the commas. Having the default information allows pagers to locate the needed fields quicker once they learn what each field stands for. Having used Xtags for over nine years, I use this setting almost exclusively.

Output separate tags (e.g., <I> vs. <BI>)

When selecting this, the program will generate separate Bold, Italic, etc. tags rather than combining tags. I usually like to have them combined, but this all has to do with your workflow and if you have defined scripts to do something during the post processing phase.

Delimiters

One standard about any programming software or language is that certain characters are reserved for certain functions. With Xtags, I find it's better to avoid the use of the following characters whenever possible:

```
# \ & : = ; . + - < >
```

Do not use these unless they possess a definite function that won't be confused with something operational.

Get Text and Save Text with Xtags

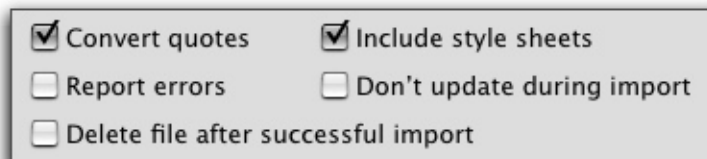
Get Text with Xtags

To import tagged text into the QuarkXPress document using Xtags, you cannot go about it with the standard "Get Text" mode that you are used to. This

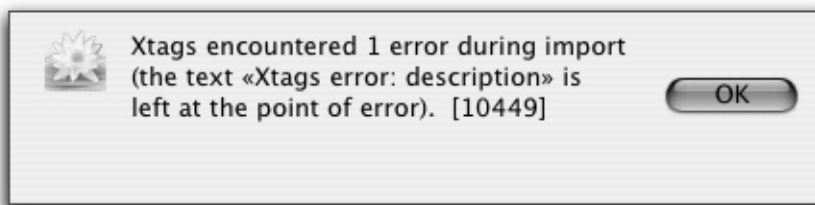
14 *Xtags Maximized*

will happen by going to *File: Get Text with Xtags*. A window will open prompting you to find the text file that you want to import.

There are 5 different buttons that can be selected. I typically use the *Convert quotes* and *Include style sheets* options. The *Convert quotes* needs to be selected if you want the (") to become the smart quote equivalent. *Include style sheets* needs to be selected for Xtags to process the text properly.



If you are fairly new to Xtags, it is a good idea to have the *Report errors* option selected so any problems with the tags can be easily located. Upon running the text, a message will prompt reporting the number of Xtags errors that are in the document.



Next to the imported text where the error resides will be an error describing what the problem is and where it is in the Xtags string:

«Xtags error: Malformed tag: tag &tbu2 param #7»

I purposely made this error to show how easy it is to find. Here's the unanchored text tag with the error:

<&tbu2(0,0,34p,12p,,Z,)>

The seventh field has an "Z" in it which is not an option for that parameter. This generated the error. At this point, go back to your file and correct, or if using a translation table, the code would need to be corrected there. It is always a good practice to do your work in the translation table. This saves repeated steps.

The *Don't update during import* checkbox can speed up the importing by not doing a live update on the screen, but rather just displaying the text once everything has been successfully imported.

Be very careful when selecting the *Delete file after successful import* checkbox. This will actually delete the entire text file. I would only suggest using this if you have a backup of the text file somewhere and you moved the text

file to a location on your hard drive. Less clean-up is required later, but can be a problem if you only have one copy of the file and deleted it. Use caution if selecting this.

Importing with Autopage

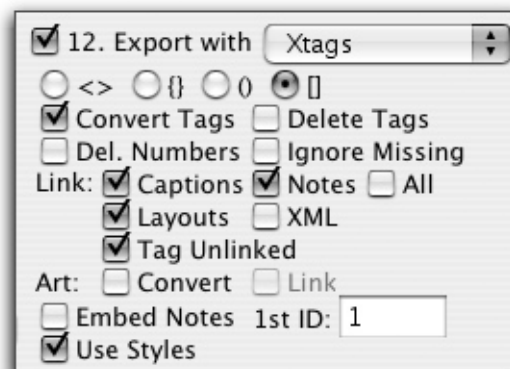
If using Autopage, the user can also go to the Autopage Paginate window and select “Import - Xtags” and get the same result as “Get Text with Xtags”. Many pagers prefer this method in their workflow.

Save Text with Xtags

To save the text of an existing Quark file with Xtags, open the Quark file, place the cursor in a story, and go to *Edit:Save Text with Xtags*. All anchored text and picture elements of that story will save out with Xtags strings as well. Any floating elements will not export which can pose a problem. This is where the *Batch Export* with Autopage is very useful.

Exporting with Autopage

If you are a user of *Autopage*, you can go to the Paginate window and through the #11 and #12 options, all of the text and floating element in all stories will export. This is a big savings of time and enhances the Xtags experience.



This also works in Autopage when using *Batch Export* where you can export the text of all chapters in a folder. This is the preferred method when using this feature. If using an XML workflow with an XTension like XMLxt, you can actually get the art to fall in the location where it was called out.

AppleScript Get and Save Text

I'm a major user of AppleScript with Xtags and it's a fairly easy process to get working with. I'm not one to waste production time while searching for files. If I know their location, I will have an AppleScript built and just change the chapter number.

16 *Xtags Maximized*

Here's an example of an AppleScript with the *Get Text with Xtags* function. I have included both *quote conversion* and *error reporting*. The *convert quotes* is the default.

```
tell application "QuarkXPress"
  activate
  get text with Xtags from "Macintosh HD:Automation
    Folder:Wexler:Wexler_ch03" with quote conversion
    and error reporting
end tell
```

If you didn't want *Error reporting*, simply write the "get text" line as follows:

```
get text with Xtags from "Macintosh HD:Automation
  Folder:Wexler:Wexler_ch03" with quote conversion
```

This will also work for the *Save Text with Xtags* function. Some people may argue that it is easy to just go into the file and save the text with Xtags and it might not be worth the time investment to write the AppleScript. This can be a good point, however, you need to look at your day and it seems to me that any time you can save from having to look for folders or annoying tasks like this only ensures more profitability to your employer. So it is a good practice to see wherever production time can be reduced.

Here is an example where I instruct the program for *style definitions*, *full list elements*, and *containing box*. When selecting containing box, Xtags will throw in a Xtags Unanchored text box tag <&tbu...> and a closing <&te> around the text. If you do not want this, don't include this in the AppleScript

```
tell application "QuarkXPress"
  activate
  save text with Xtags to "Macintosh HD:Automation
    Folder:Wexler:Wexler_ch03.txt" with full list
    elements, containing box and style definitions
end tell
```

The time saver here is that I already have the file named, how I want the elements saved, and the location it will be saved in.

Copy and Paste Xtags

Copy Xtags Text

Similar to the *Save Text with Xtags* is the *Copy Xtags Text* option. I use this heavily when figuring out my boxes, localized style overrides, and anchored

text. I can usually write these out because I'm so familiar with the program, but I suggest that this is where you start. It leaves a lot less to chance.

Let's take this example below. It's a very basic example of a numbered list where the number requires a 20% gray circle. The number needs to reside within the circle.

2 See the previous figure

By seeing this example, I would do the following to figure out the proper kerning, baseline, shift, etc.

1. After getting this set up properly, select the text from the start of the circle to the word "See". Make sure you have a character style applied to the bullet.
2. Go to *Edit:Copy Xtags Text*.
3. Either make a text box off to the side, or open BBEdit. Paste the copied text into the file. It will read as:

```
@SNL1:<@SNL_Bullet><k-60>l<@$p><b0.5>4<\f><\f><b0>See
```

4. This is exactly what you want. You can now achieve this by putting two of these pieces in the translation table as follows:

```
[[sn]]    <@SNL_Bullet><k-60>l<@$p><b0.5>
```

```
[[sn2]]   <\f><\f><b0>
```

5. The markup department would then have to type these when coding the text file as:

```
@SNL1: [[sn]]2 [[sn2]] See the previous figure.
```

6. This could also be added automatically by having the coding department type the number followed by a tab, and through a Perl Script you could add this line:

```
s/(\@SNL1:|\@SNL:)(\d+)(\t)/$1\[\[sn]]$2\[\[sn2]]/g;
```

This is a simple example and I will go into more detail on how to use the *Copy Xtags Text/Paste* combination in later chapters. I find this to be very useful when trying to write or adjust the codes for figures and boxes.

Paste Xtags Text

The other option is to *Paste Xtags Text*. This works after you've either completed a *Copy Xtags Text* to work with an element and then after some code adjustments you want to see how it looks. This is where *Paste Xtags Text* comes into play.

18 *Xtags Maximized*

We'll use the same text from the example above to show how this would benefit you.

1. In BBEdit or on the pasteboard, the code is written that you want to use in your translation table. Copy the entire string of text.

```
@SNL1:<@SNL_Bullet><k-60>l<@$p><b0.5>6<\f><\f><b0>See
```

2. Select a text box or a place in the Quark file that you want to paste this object or text string. Go to *Edit:Paste Xtags Text*.

6 See

3. This will display the text to see if any adjustments need to be made.
4. You can also *Copy* and *Paste Xtags Text* the entire text file instead of using *Get Text with Xtags* if you so desire.

My goal here was to get you familiar with these concepts since they will be mentioned in more detail later throughout other chapters.

Style Sheet Tag Attributes

One of the features that I work with in great detail is the Paragraph and Character Style Sheet Tag Attributes. Many users I know don't pay enough attention to these, but a whole new world can open up if you take these and find quicker ways to improve your current workflow.

One of the most incredible features of Xtags is in its ability to define a style sheet before it's defined in QuarkXPress. The style sheet definition appears as follows:

```
@h3=[S","h3"]< *L>< *h"P3">< *kn0>< *kt(2,2)>< *ra(1,"Solid",K,60,0,0,9)>< *rb0>< *d0>< *p(0,0,0,14,0,0,g,"U.S. English")>< *t(18,0,"1 ")><P><s100><t0><h100><z12><k0><b0><cK><f"Bembo-Bold">
```

This is the way a style sheet is created from scratch. If you were to copy the text above and then select *Paste Xtags Text* into a new document, this style sheet would be defined above. Two parts of this would not work properly if you didn't have these figured out ahead of time.

1. The first would be the H&J if the "Pre P3" wasn't defined. Then it would just default to "Standard". It is always best to have your H&Js ready to go ahead of time. I will show you a good AppleScript way to do this later in this chapter.
2. The other is to make sure the font is loaded and that you named it properly in the tag. Otherwise, it will become the default font in the normal style sheet.

By using this way of defining a style sheet, a person can quickly reduce the time it takes to design a book. I have built a FileMaker Pro database that can actually speed up the design time on a book by using pull down menus to populate these tags. Omitted tags in the style sheet default to the “Normal” style attributes.

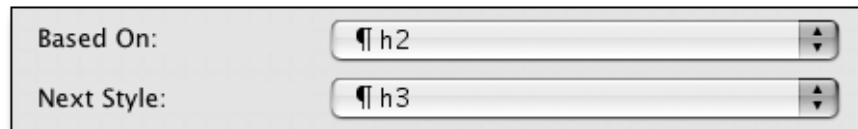
I’ll break down each of these tags, so you can see how these work together to form a full style sheet. Be aware that spaces are allowed within the tags and they will still work the same. For example `<*L>` is still acceptable as `< * L >`.

Style Sheet Definition

`@h3=[S"", "h3"]` Calls in the style sheet

`@h3=[S"h2", "h3"]` Calls in the style sheet and the style based on

The top example creates a style sheet named “h3”. The bottom example shows an “h2” in the second set of quotes meaning the “h3” style is based on the “h2” style. The “h2” style would need to be defined before it any style can be based on it. Here’s how it would look in the format window:



Paragraph Alignment

`<*L>`

This is the alignment of the paragraph. The breakdown for the alignment is as follows:

<code><*L></code>	Left alignment	<code><*J></code>	Justified
<code><*C></code>	Center alignment	<code><*F></code>	Forced Justified
<code><*R></code>	Right alignment		

Hyphenation and Justification (H&Js)

`<*h"Pre P3">`

This is where the H&J is defined. It is very important to have these predefined. Any name can be put in between here, but if it doesn’t reside in your H&J list before importing the text, it will default to “Standard”. Therefore, the best solution is to either have a template that is already created consisting of all of your H&Js, or to have a AppleScript that you can run against your Quark document that will put these in for you ahead of time. I find this to be

20 *Xtags Maximized*

the best practice. A lot of times you are dealing with another company's files and want to have your H&Js incorporated into those files.

I actually have an AppleScript that I created that is basically a quality assurance checker. It creates the H&Js, fixes the colors, and has my preferences just how I want them. Here's an example of how I would write the AppleScript for the H&Js:

```
tell application "QuarkXPress"
  activate
  make new h and j spec at beginning of document 1 with
    make new h and j spec at beginning of document 1 with
      properties {name:"H1", auto hyphenation:false, single
        word justify:true}
  make new h and j spec at beginning of document 1 with
    properties {name:"Post P1", auto hyphenation:true, minimum
      before:3, minimum after:3, break capitalized words:false,
      flush zone:"p3", single word justify:true, space
      justification:{minimum:"85%", maximum:"200%",
        optimum:"110%"}, character justification:{minimum:"0%",
          maximum:"3%", optimum:"0%"}}
end tell
```

This AppleScript just shows how to write it that will work. To add more, just change the names that are highlighted. The “H1” has no justification and the “Post P1” has hyphenation.

Keep with Next

<*kn0>

Keep with Next has only two selections. Either <*kn0> which means the *Keep with Next* is unselected, or <*kn1> which means it is selected. Having the <*kn0> is great to incorporate on paragraphs in Autopage where you do not want an inline paragraph to be with the next. A “Grep” or MacPerl replacement could be run against the text that could specify:

```
s/^(^+?)(\[\.+?I=Y)/$1<*kn0>$2/g;
```

This is telling the paragraph that if it sees the “I=Y” Autopage tag to remove the *Keep with Next* selection on that paragraph.

Keep Lines Together

<*kt(2,2)>

Keep Lines Together is extremely important to the paged file. The above example is showing that this feature has the “Start 2 End 2” selected. If you have

this deselected, the pages will sometimes have some very unfavorable results. There will be times when a single line will sit on at the bottom of the page after a head. This is not acceptable for the majority of textbooks. I make it a habit to have the “Start 2 End 2” selected. Here are the different options:

`<*kt0>` Keep Lines Together (deselected)
`<*kt(2,2)>` Start 2 End 2 Selected
`<*kt(A)>` All Lines in Paragraph

The “All Lines in Paragraph” together is something I normally don’t get in the habit of doing because it can give unpredictable results where big blocks of text end up together. It works well if you have a small box of a few lines that you do not want breaking across pages. For this option, I try to stick to floating elements or heads with multiple lines.

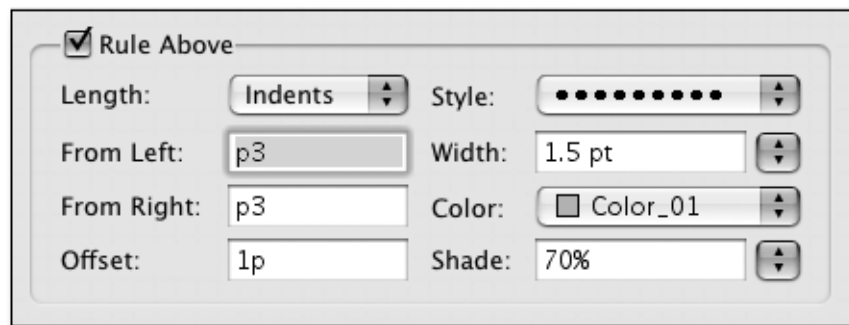
Rule Above and Below

`<*ra(0.5,"Solid",K,100,0,0,11)>`

The *Rule Above* and *Rule Below* has a lot of possibilities in one tag. The *Rule Above* is `*ra`; a *Rule Below* is `*rb`. If not defining a rule above or below, a simple `<*ra0>` and `<*rb0>` tag is inserted. Rules added to override defined style sheet settings are necessary when paging. The structure for the rule tag is broken down as follows:

(thickness, style, color, shade, left indent, right indent, offset)

Most of these are self-explanatory by looking at the *Rules* “Paragraph Attributes” window in QuarkXPress where all of the elements are specified except whether the length is “Indents” or “Text”.



The default is “Indents”, but if you select “Text”, then right before the Left Indent, a “T” must precede it. Here’s an example of how this would appear:

`<*ra(0.5,"All Dots","Color_01",100,T0,0,18)>`

22 *Xtags Maximized*

As seen on the previous page, you must name the rule style in quotes. These need to be named appropriately such as “All Dots”, “Thick-Thin-Thick”, etc.

The color name must be in quotes. The only exceptions to this directive is if the color is Cyan (C), White (W), Black (K), Magenta (M), or Yellow (Y).

I’ve found that there are many instances when a closing rule is necessary to locally style and not have added to the existing style sheet. For example, certain boxes need a closing rule that cannot really be added to the style because the last style of the box may end with an extract, display math, or paragraph. This also cuts down on the multitude of styles that are necessary. The best scenario in this case is to have an override code added on the last style of the box regardless of what it is:

```
<*rb(1,"Solid",K,100,0,0,6)>
```

If any style falls at the end of the box, the rule will fall below. An issue that may occur is if any of the styles are indented, such as an “extract”. In this case, you would incorporate:

```
<*rb(1,"Solid",K,100,-12,0,6)>
```

This would be on a case-by-case basis. These rules can be added easily in a translation table by just using the codes `[[rule]]` and `[[rule2]]`:

```
@BOX_T1:[[rule]]This is the closing paragraph.
```

```
@BOX_EXT:[[rule2]]This is the closing paragraph.
```

With the aid of “Grep” or MacPerl, it is very easy to change this based on the style name preceding it, but there would have to be a way for it to know the box was ending. The markup/coding department could put a code `[[boxend]]` at the end of the box that could be swapped out. Here’s a MacPerl example:

```
s/(\@BOX_.+?:)(.+?)(\\[boxend]])/$1\\[rule]]$2/g;
```

I’m having MacPerl incorporate the macro for the translation table. In order to add the `[[rule2]]` for the BOX_EXT style you must place this second in the order of the script:

```
$two = “2”;
```

```
s/(\@BOX_EXT:)(\\[rule])([ ])/$1$2$two$3/g;
```

Drop Caps

```
<*d0>
```

The *Drop Cap* tag isn’t used as much as usual and typically defaults to the `<*d0>` tag. If you do need to have a drop cap defined, the tag looks like this:

```
<*d(1,2)> Drop Cap: Character Count 1 Line Count 2
```

Paragraph Parameters

```
<*p(0,0,0,14,0,0,g,"U.S. English")>
```

The paragraph parameters tag has a lot of fields to enter. I'll break this down easier by going left to right:

(left indent, first line, right indent, leading, space
before, space after, lock to baseline grid, language)>

I typically leave the "lock to baseline grid" and the "language" alone. This will reflect the formats window in QuarkXPress. If you were working with a numbered list, the paragraph parameter may reflect this:

```
<*p(18,18,0,13,9,0,g,"U.S. English")>
```

You can also use picas in the left and right indents as well as the first line. Getting to know this tag can add functionality if you plan on making a style database or a script to change existing styles.

Tabs

```
<*t(18,0,"1 ")>
```

The *Tabs* will not always appear in the full style definition by default. If you are writing your own definitions and you want to ensure no tabs will be present, enter <*t0>.

Tabs are defined as follows:

(position, alignment, fill)

The *position* is the ruler position where the tab sits. The alignment is which type of tab it takes. Here are the alignment definitions:

0	Left alignment	4	Decimal
1	Center alignment	5	Comma
2	Right alignment	"?"	Align On

The *fill* character is a little more involved. I'll try to explain this as I understand it to be. The fill isn't always going to be used. In fact, many times you won't want a fill in your tab. By leaving this blank, a "1 " will appear. However, if you want to have a single digit such as a pattern, you would type in a "1.." and this will give you the following appearance with the tab:

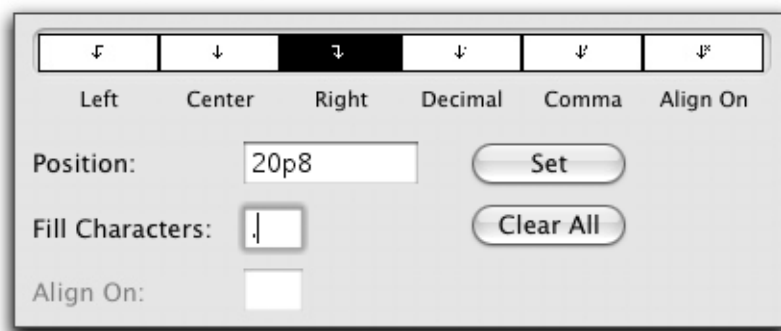
Price of 3 books \$49.95

If you would like the dots to have a space between, it would need to read as "2 ." and the tab will give this appearance:

Price of 3 books \$49.95

24 Xtags Maximized

By looking at the *Tabs* window in QuarkXPress, it is easier to make sense of this tag.



This particular tag would be written as

```
<*t(20p8,2,"1..")>
```

The breakdown for this is that the Position is 20p8, the Right tab is “2”, and because the Fill Character is just a “.”, it gets the “1..” indicator. When using the “Align On” command, the tag would contain the character you wish to align on. For example, if you wanted to align on a “;”, the tag would read as follows:

```
<*t(20p8,";", "2.")>
```

Two tabs in one tag will need to be separated by a comma and will appear as follows:

```
<*t(43,0,"1 ",175.288,0,"1 ")>
```

Character Information in the Style Sheet Definition

Type Style

<P> This is the *Type Style* enhancement that is in the definition. It will either fall right after the tabs or paragraph information. The options are as follows:

<P> Plain	</> Strike-through
 Bold	<K> All caps
<I> Italic	<H> Small caps
<O> Outline	<+> Superscript
<S> Shadow	<-> Subscript
<U> Underline	<V> Superior
<W> Word underline	<\$> Back to current style

These can be used in the styles or just used in paragraphs for local styling. I find when using Bold or Italic, it should be set in the actual font. For example, if you want an Times Italic. Make sure you are using “I Times Italic” opposed to “Times” with <I> in the style sheet. The enhancements can end with the same character or with a <\$>. Here is a good use of one of these tags in the actual text flow when styling a word needing italicized. I’ll show both acceptable endings.

<I>Figure 1.1<\$> or <I>Figure 1.1<I>

Shading

<S100> This is the *Shading* of the type color. The range is from 0-100.

Tracking

<t0> This is the *Tracking* of the type. Do not get this confused with the <*t0> tab code. There is no asterisk in the character styles. The range is from -500 through 500.

Horizontal and Vertical Scale

<h100> This is the *Horizontal* or *Vertical Scale*. Only one or the other can be used. *Horizontal* scaling uses the <h100> code. *Vertical* scaling uses the <y100> code. The range is 25 through 400.

Type Size

<z12> The “z” code represents the *Size* of the type.

Kerning

<k0> This is the *Kerning* of the type. The only problem I’ve found with setting this in the style sheet definition is that there is nowhere to control it in the style sheet. If necessary, use it. The tracking can work similar if you play with it and find the proper look. The range is from -500 through 500.

Baseline Shift

<b0> This is used for baseline shifting. It can be positive or negative. The range depends on the size of the type.

Color

<cK> To add a custom color into an Xtags string, it is critical to have the color already created in the QuarkXPress document/template. If the color is not included, Xtags will substitute the defined “normal” style color for the missing color; however, it will not add it to the color definitions. If the *Report Errors* is clicked, it will generate this error:

«Xtags error: No such color: tag c»

Coloring type with Xtags is as simple as having the color defined in the document and then typing the following tags:

```
<c"Color A">words needing colored<cK>
```

The <cK> returns the type to black.

Font Definition

<f"Font"> This is where the font definition goes. It is very important to have this defined correctly or the font will default to the font used in the "Normal" style sheet. For example, if the style sheet needed <f"Bembo-Bold"> and you typed in <f"Bembo-Bold"> with an en-dash between it, it would take on the Normal style of type. It is very important to make sure the name is correct. Test this first.

Creating Character Styles

Very similar to creating a Paragraph Style, Character Styles use only the last half of the style tag from the type style to the font definition. Here's an example of how the character style will appear:

```
@H4=<K><s90><t0><h80><z11.5><k0><b0><c"C10"><f"Berkeley-Bold">
```

The descriptions on the previous pages apply exactly the same to the character styles. When defining a character style, it is possible to only include the important information. For example, you could just enter the following:

```
@H4=<K><s90><h80><z11.5><c"C10"><f"Berkeley-Bold">
```

The undefined tags take on the "Normal" style specifications.

Style Sheet Overrides

If you need to override a style sheet, you can use whatever tags you would need to make this happen. For example, if a workbook needs a different setting for one list element, but a style sheet isn't necessary, it is possible to put this type of a treatment in the style sheet:

```
@BXNL:<*p(1p10,-1p10)><*t(1p10,0,"1 ")>1.<\#009>This is the type.
```

As shown above, the paragraph parameters that normally contains 8 fields looking like <*p(0,0,0,14,0,0,g,"U.S. English")> has been broken down to only needing the first two. This is acceptable. If you needed to just change the leading, it could be written as:

```
<*p(,,18,,)>
```

All of the other non-specified fields will take on the style sheet's attributes.

Paragraph Style Sheet Examples

Just in case the paragraph and character style definitions need a little more verification, I thought I would spend the next couple of pages showing examples of how building these styles can save you production time.

Example 1: Drop Shadow on Head

Let's say you wanted to have a drop shadow on a head that looks like this:

H1 Head Example

Typically most people would take the shadowed "H1 Head Example" and put the shadow in a different text box and try to align it by hand or by the *x* and *y* coordinates. This requires additional work. It could be handled this way, but it will only make the design that much harder to deal with. If using Autopage, it would tend to be very difficult. The following steps are much easier:

1. Bring in the H1 head twice. The type size on this example is 16/12. The indent is p7. *Keep with Next* will need to be selected for the top H1 head. I shaded the bottom example for visibility only.

H1 Head Example

The "H1_Shade" style is the first line that comes in since it will be the shadow. Pay attention to the highlighted tags. Here are the definitions:

```
@H1_SHADE=[S","H1_SHADE"]<*L><*h"4"><*kn1><*kt0><*ra0>
<*rb0><*d0><*p(7,0,6,12,0,0,g,"U.S. English")><P><s100><t0>
<h100><z16><k0><b0><cK><f"Futura-Bold">
```

2. The duplicate H1 head will need to have 2 points leading with a baseline shift of 3. The color needs to be white. The left indent is set at only p6 on the duplicate head. This will give the slight offset. I've highlighted those tags that need to be changed compared to the original

```
@H1=[S","H1"]<*L><*h"4"><*kn0><*kt0><*ra0><*rb0><*d0>
<*p(6,0,7,2,0,0,g,"U.S. English")><P><s100><t0><h100>
<z16><k0><b3><cW><f"Futura-Bold">
```

3. The two of them will now come together with a slight offset. This is much quicker than setting it up as side art or adding a second box. Use the power of the Xtags style sheets for creating faster workarounds like this.

End Result:

H1 Head Example

Example 2: Screen behind a box head**MAKING A CHECKLIST**

Certain projects will need to vary from one to the next, however, it is important to come up with standards. Look at a previous book and start there.

Here is a good example of a screen behind a box head. When setting up the Xtags boxes, I've observed people putting the screen as its own separate box. This is not necessary. Although Xtags has the power to do things like this with picture box tags, I feel it is best to keep this simple and defined in Quark instead. A style sheet can be created ahead of time to handle this.

This can be done in QuarkXPress style sheets or through Xtags by using a rule above and rule below. Here is the Xtags handling of this:

```
@BXHD=[S","BXHD"]<*C><*h"4"><*kn0><*kt0>
<*ra(21,"Solid",K,20,0,0,-5)><*rb(0.5,"Solid",K,100,0,0,7)>
<*d0><*p(0,0,0,15,0,9,g,"U.S. English")><K><s100><t0>
<h100><z12><k0><b0><cK><f"Futura-Bold">
```

Example 3: Mathematical Overbar

With math programs available like *MathType*, it is very difficult to work with math equations the old fashioned way. However, some people do not work enough with math books to make it worth the investment. Using Xtags and overriding the paragraph style can lend a hand in creating these and coding them as well. A good example is the need for an overbar as seen here:

$$\overline{15}$$

The code for this will be as follows:

```
@EQ:1<k-193>5<k0><b7.4><\m>
```

The kerning would need to be -193 and the baseline shift before the em-dash would be 7.4. This will give this effect. The great part of this is it can be then broken down in the translation table into two parts to save coding time. You could have

```
[[eq]]      <k-193>
[[eq2]]     <k0><b7.4>
```

The tagged text file would reflect this on any two digit number:

```
@EQ:2[[eq]]7[[eq2]]<\m>
```

The result would be:

$$\overline{29}$$

Example 4: *Items behind type*

There are many different situations in paging where the designers will decide to set something up a certain way that is easiest for them, yet it isn't efficient when paging. Items sitting behind the type or heads are good examples of this. Here is an instance of an oval underlaying the start of a head.

Using Boxes and Elements

Many times when using a program like Autopage, the pager will just use the oval as side art, and underlay the head with the oval. Unless the oval was sticking out into the margin area, I wouldn't handle this that way. As a pager, setup person, automation specialist, or whatever your title is, you need to find efficiencies. It's imperative to the success of your projects. Don't be afraid to be creative with dingbats. In this case I will show you step-by-step how I would approach this. I'm certainly not breaking any new ground here, but it can be interesting how people don't think of these things.

1. The first thing I do is bring in a circle with Zapf Dingbats. I apply a 180 horizontal scaling and 25% shading. Now we have an oval.

Before Styling  After Styling 

2. When imported, the oval and the head would appear like this:

Using Boxes and Elements

3. By tracking -60 between the dingbat and the letter "U", we get the desired effect.

Using Boxes and Elements

To code this through Xtags using the oval in a character style sheet, the code would be:

```
@H1:<@H1_Oval><k-60>l<@p>Using Boxes and Elements
```

Don't be afraid to experiment with the dingbat fonts to get the effect before you start working with floating elements. The next example will show something similar with bullets.

Example 5: *Rectangle numbered bullets*

This is something I see many times in construction or plumbing books where multiple steps are numbered. They usually have each step numbered, but the designer gets creative and does something like this:

1 This is the first entry.

This can be handled two different ways and both are equally effective. The first possibility is that it could be a stretched Zapf Dingbat square with a

30 *Xtags Maximized*

negative tracked number, or it can be set in the rules format by using tabs on the number.

Let's first look at the stretched Zapf Dingbats. This is probably the easiest way to approach this.

1. We'll need to bring in a square "n" Zapf Dingbats, and this will need to be horizontal scaled at 190% with a baseline shift of -2.



2. Next, we'll bring in the number on the side of this. For demonstration purposes, I'll shade this 40%, so it's visible, but normally this will be white. The number will need to be tracked back -52 to fit perfectly inside the rectangle. Double digits would need more tracking.

Before Tracking  1 After Tracking 

3. The Xtags to import this (besides changing the number) would be:

```
@BL:<@BL_DG><k-52>n<@BL_NM>l<@$p>
```

The other method is to set the rectangle in the style sheet with the rules where there would be a negative offset to bring the rule above back. Here is the Xtags paragraph rule style for this:

```
<*ra(12,"Solid",K,100,1,483,-0.5)>
```

This would require that a center tab is set so the number will position in the middle of the rectangle.

 2 This is second line of type

Example 6: *Square bullets set with rules*

When working with square bullets in text, it is just as easy to not have these coded for and set them in your style sheet in the rules.

- This is the bulleted element
- Here is the next bulleted element

Here is the code in the Xtags paragraph style sheet of how this. The indent from the right is 19p6.

```
<*ra(6,"Solid",K,100,0,19p6,0)*>
```

That is how this sets in perfectly without any additional coding. The coded file for this would be:

```
@BL:This is the bulleted element
@BL:Here's the next bulleted element
```

The key here is to look for any advantages when paging that will save the markup department from additional coding to simplify the paging.

Xtags Export Issue

When exporting the style sheet information, I have noticed with Xtags 6.2 that the tags are no longer separated like they were in the Xtags 4.2 version. Here's an example of the Xtags 4.2 version:

```
@h3=[S","h3"]<*L><*h"P3"><*kn0><*kt0><*ra0><*rb0><*d0>
<*p(0,0,0,14,0,0,g,"U.S. English")><P><s100><t0><h100><z12>
<k0><b0><cK><f"Bembo-Bold">
```

Here's a style tag in the Xtags 6.2.1 version:

```
@H5=[S","H5"]<*L*h"P3"*kn0*kt0*ra0*rb0*d0*p(0,0,0,14,12,6,g,"U.S.
English")Ps100t0h110.001z12k0b0cKf"Futura-Condensed">
```

I don't want to make too big of a deal of this because it's very possible that they will fix this in an upcoming version, or they changed this for a reason...possibly to do with the possible conflicting of XML tags. I personally like working with the <*kn0> type of tag rather than trying to make sense of the long string. I have devised an AppleScript workaround to fix this issue.

What I do is by looking at the tags, I can decipher what needs to be done to each one in order by the options. Let's start at the beginning. For the <*L tag, I will need a replacement that changes this. My "Grep" changes will be:

(*)(L R C F J)	replaced with	& >
(*)(h)(\")(.(?)(\")(replaced with	& >
(*)(kn)(.(?)(*)	replaced with	\1\2\3>\4

Here's an example how the AppleScript would appear. This would be using "Grep" substitutions in BBEdit. Be sure you have the *Case Sensitive* selected as "True" or you could have problems.

```
tell application "BBEdit"
    activate
    replace "(\*)(L|R|C|F|J)" using "&>" searching in text 1 of text
        document 1 options {search mode:grep, starting at top:true,
        wrap around:false, backwards:false, case sensitive:true,
        match words:false, extend selection:false}
    replace "(\*)(h)(\")(.(?)(\")(

```

By running this script, it will take these tags below:

```
@H5=[S","H5"]<*L*h"P3"*kn0*
```

And it will become the following:

```
@H5=[S","H5"]<*L><*h"P3"><*kn0*>
```

This is just something that you can mess around with if you are having trouble deciphering the tags if they remain how they are now.

Changing the Inset (for Autopage 5.8)

This particular description only applies to Autopage 5.8 users (System 9.2) users because this has been rectified in Quark 6.5. In Quark 4.11 or earlier, the inset value could only be created with one value for the entire inset when using *Copy Xtags Text*. There wasn't a *Multiple Insets* as there is in Quark 6.5. However, that does not stop one from being able to set this up in Xtags.

A good example of this is requiring a simple box with a rule around it, but needing a (12,12,9.5,12) inset. During a *Copy Xtags Text*, the inset would paste as:

```
<&tbu2(0 B,0,20p,25p?,,,,n,0.5,(n),(,100),,n,,,,,12,,,,,)>
```

This can be manually changed to:

```
<&tbu2(0 B,0,20p,25p?,,,,n,0.5,(n),(,100),,n,,,,,(12,12,9.5,12),,,,,) >
```

This will then give you the inset that is needed. Whenever possible, you need to look at ways for Xtags to fix issues like this.

French Quotes vs. Brackets

Many pagers use *French Quotes* (« ») for their macros when setting up their translation table. *French Quotes* need <\#199> and <\#200> ASCII codes, so they have to be first imported into a QuarkXPress file, then the file needs to be resaved as ASCII before using *Get Text with Xtags*. That's a waste of time. I suggest using the same delimiters used in Autopage: [[and]]. As long as your translation table is set up like this, you'll have no problem with these as long as you never name your tags in the same structure as AutoTags.

Translation Tables

A properly set-up translation table can have the same power as an intricate AppleScript or MacPerl script. I've always used the translation table as much as possible for importing art, writing Autopage codes, creating boxes, style overrides, and so on. It is immeasurable the amount of time a translation table will save you from doing repetitive tasks.

The idea of the translation table is to take your complex strings and put them in one place while a small macro identifies the string in the input text file. During the Xtags import process, these macros take the information within the translation table and use the longer string. Typists and coders do not want to have to type these long strings in each time they occur.

For example, if using a program like Autopage, you may have a layout change that occurs at every H1 head.

```
@H1:[[LC 1 M=30p A=4 S=.5 H="Cyan"]]  
The H1 Head
```

Instead of coding this each time at the @H1, this string of code can be put into a translation table with a small macro. I will call this layout change [[LCA]]. In my translation table, the following will appear

```
[[LCA]]      [[LC 1 M=30p A=4 S=.5 H="Cyan"]]      L_Change A
```

The translation table is broken down as:

- 1st field** – The name of the tag/macro/source
- 2nd field** – The string/target
- 3rd field** – Comments (optional)

Each of these fields become delimited by using a tab between each field. It's extremely important to use tabs or formats if you want to align objects in the translation table. Do not use *hangs* (Command \) to align objects or the translation table will not work properly.

Similar to most scripting languages, the translation table can have lines that are commented out so you can make notes. To do this, the first character in the document defines what the comment character needs to be. Out of habit, I always use the “;” because I rarely see this used anywhere.

To comment any line, this character must be placed before the line you want to write notes or comments on. The example at the top of the next page shows how an Xtags translation table written in BBEdit should appear.

This translation table shows 2 types of figures, tables, and a few other items thrown in including a layout change. I've actually had medical books that were so involved that my translation table exceeded 4 pages. That is uncommon, but it proves how critical it is to understand how this works.

Translation Table Start Tag

Whenever a file needs to be imported, a tag has to start the text file in order for the file to import the translation table information. If the file name is “book.ttl”, the tag at the beginning of the text file will need to read:

```
<&tt2”book.ttl”>
```

Using this tag requires that the translation table be saved in the folder with the template. If working on a server, this allows anyone to be able to work

```

ζautopagebook.ttl
ζtranslation table for autopagebook xtags

ζfigures
[[f1]]      <&pbu2(0 B,0,28p?,48p4?,0,0,,n,0,(K,W),(100,100),"Solid",K,0,,c,100,100,0,0,0,0,"
[[f2]]      ",," "><&tbu2(0 BL1,10.5,28p,6p6?,0,0,,n,0,(K,W),(100,100),"Solid",n,0,0,1,0,0,0,,t,0," ">
[[f3]]      <&te><&g(2,1)>

ζtables
[[t1]]      <&tbu2(0,0,30p,45p?,0,0,,n,0,(K,W),(100,100),"Solid",W,0,0,1,0,0,0,a,t,0," ">
[[t2]]      <&te>

ζfigures w/a left upper caption
[[f4]]      <&pbu2(14p B,0,30p?,46p4?,0,0,,n,0,(K,W),(100,100),"Solid",K,0,,m,100,100,0,0,0,0,"
[[f5]]      ",," "><&tbu2(-13p TL1,0,12p,6p6?,0,0,,n,0,(K,W),(100,100),"Solid",n,0,0,1,0,0,0,,t,0," ">
[[f7]]      <&pbu2(0 B,0,34p,48p4?,0,0,,n,0,(K,W),(100,100),"Solid",K,0,,c,100,100,0,0,0,0,"
[[f8]]      ",," "><&tbu2(0 BL1,10.5,34p,6p6?,0,0,,n,0,(K,W),(100,100),"Solid",n,0,0,1,0,0,0,,t,0," ">

ζstarting tag
[[s]]       [{<\<
[[s2]]      >}]

ζending tag
[[e]]       [{<\</
[[e2]]      >}]

ζending tag
[[i]]       [{<\</
[[i2]]      />}]

ζlayout change
[[LCA]]     [[LC 1 M=30p A=4 S=.5 H="Cyan"]]      Layout Change A

```

with this file. For archiving, it's much easier to have all the items together. Keeping it in the same folder as the template is similar to OS X putting everything in your user folder.

The translation table can also be saved in the *Xtags Preferences* within the *Library* of the *System* as well, but this requires a longer string to find the parameters and it's limited to your machine.

Start with a Template

I feel that it is always a good idea to have a translation table template somewhere on your server that you would use at the beginning of any project. It is best to use this as a shell. You may even have a couple figure options in it.

I find that this speeds up the process and keeps you from having repeated steps and errors.

The best start is to have a figure, caption, and the grouping code in. Here is an example of where I would normally start a translation table with:

```

;bookname.ttl

[[art1]]      <&pbu2(0 B,0,35p?,55p?,,,,,n,,(,n),(,100),,K,0,,m,,,,,"56 GB
               Disk:Xtags Book:

[[art2]]      ",,,)><&tbu2(0 BL1,12,35p,5p?,,,,,n,,(,n),(,100),,n,0,,,,,)>

[[art3]]      <&te><&g(1,2)>

```

The highlighted areas are where you would have to adjust in most cases. Even though we haven't covered the picture and text box tags, I will at least touch on this quickly. You would want the maximum width and depth of the art to change. The main changes will be the spacing between the art and caption, and the path to the book.

As you can see, the path to the art is not complete. This is where I'd have a database like FileMaker Pro add the images, or have an AppleScript do this. There are many ways to accomplish this including having a markup department just type it in. I find it the best to use scripting whenever possible and starting with a short code marker. We will discuss this in a later chapter. The code then will look like this in the final coded file:

```

@fgc:[[art1]]Chapter 01:fg01_001.eps[[art2]]<@fgn>Figure
1.1<\f><\f><@$p>Figure Caption[[art3]]

```

Translation Table Codes

For certain characters, it's important to use the ASCII coding for these to work. The following codes need to be used within a translation table to work properly when imported:

```

Tab      =    <\#009>
Return   =    <\#013>
Backslash =    \\

```

Translation Table Format

A translation table needs to be an ASCII file that can be set up in most applications, but I prefer using BBEdit. I refer a lot to BBEdit and I think it's one of those programs that no pager should be without. The time saved alone on working in the translation table is worth the investment. It's best to just start in a text editor and save with the .ttl extension on the end of it.

It is also possible to set up the translation table in Quark and then do *Save Text* and select the “ASCII Text” format. This requires an extra step and additional back and forth time when changes are needed. I’ve actually witnessed one setup person who would make the changes, and then he would forget to resave the translation table to ASCII again. Then he would fight the translation table for a half an hour wondering why nothing was happening. I instructed this person to work with BBEdit instead. After enough situations like this, he finally took my advice and he has had a better experience since.

Translation Table Issue

When setting up the translation table, be very careful that you do not end up with a line with just one character space on it. The text will pull in with all of the words running into each other. Here is an example of how the text pulled in when this happened:

A recent addition that was made to the 5.8 version was the ability to export the text back to an .rtf document. This is a great feature that gives publishers the ability to return the word files.

Once the single character space was removed:

A recent addition that was made to the 5.8 version was the ability to export the text back to an .rtf document. This is a great feature that gives publishers the ability to return the word files.

This can happen accidentally by just moving too quickly through the translation table. If you happen to see text importing this way, backtrack through the translation table and look for a single-space character sitting on its own line, remove it, and resave the .ttl file.

Test the Codes Before the Translation Table

I’m going to mention a serious error that many setup people are guilty of. It is not testing the codes before putting them in their translation table. They do a *Copy Xtags Text* and *Paste*. Then because it came in that way, they believe it is correct. This is a bad assumption. What can happen when you are doing a *Copy Xtags Text* is that the tags will work on that particular box, but it isn’t taking into account any shrink-to-fit capabilities, relative placement, and other important aspects to make the object ready for paging.

Once the box is brought in during the import, it will remain the same size as the original tested box and you would have to manually adjust all of these during the paging process. It is always best to test these with new text and then separate the different text and picture box strings into what will be your translation table. This will save you time during the paging process. Spending a few minutes ahead of time to really test your file may save you or the pager hours of frustration later.

Master Pages

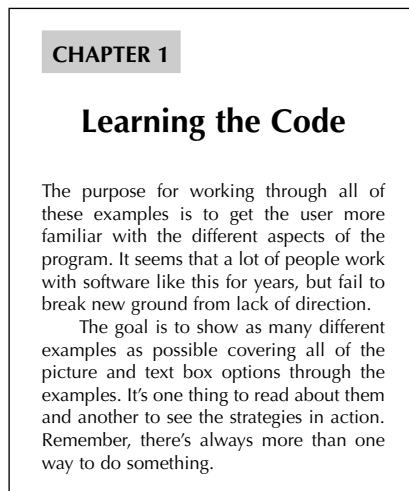
Xtags has the ability to bring in different Master Pages when called out in the text flow. This is particularly useful during the import of the text to save manual handwork. There are four supported tags for Master Pages. These are:

<code><&m"masterpage"></code>	apply to current page/spread
<code><&mf"masterpage"></code>	apply to current page/spread, first
<code><&mp"masterpage"></code>	apply to current page only
<code><&mpf"masterpage"></code>	apply to current page only, first

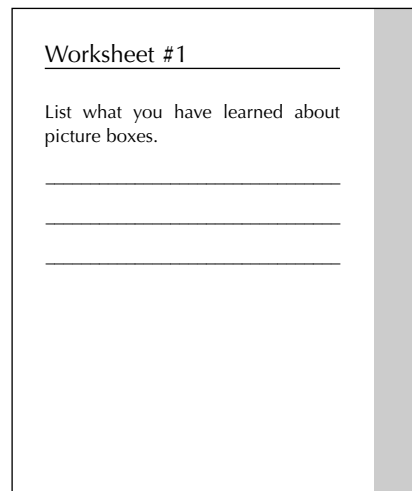
On certain projects I discovered a multitude of uses for the Master Page tags. Workbooks are a good candidate because there is little art and they tend to flow fairly well. They usually have some end matter pages requiring a certain treatment. By breaking the pages using the `<\b:>` tag, I place `<\b:>` directly before the style that falls on that page calling in the Master Page. For example:

```
@CN:Chapter 1
@CT:Learning the Code
@TX1:The ... always more than one way to do something.<\b:
><&mpf"2">@WKST:Worksheet #1
@WK:List what you have learned about picture boxes.
@WKL:<\#009>
@WKL:<\#009>
@WKL:<\#009>
@WKL:<\#009>
```

Here are the first two pages. The Master Page “2” which is “Master Page B” is called out right before “@WKST:Worksheet #1”.



Master Page A



Master Page B

The `<\b>` forced return tag keeps all of the preceding text on the “A Master Page”. The text following is then put on the “2” Master Page which is “B Master Page”. You can also call out actual titles that are put in the Master Page. For example, if you named Master Page B “Rules”, then it would be acceptable to name this Master Page:

`<&m"Rules">`

There are many uses for calling out the Master Page tags. These include:

1. Going from a one-column layout to two-columns.
2. Calling in an end matter page with a decorative bar bleeding on the outside of the page.
3. Calling in a worksheet with a different margin width
4. Calling in a Master Page with a drop folio rather than a running head.

Since most of my paging is actually done using Autopage, I rarely use this feature unless I'm paging a book that has very small chapters of 4 pages or less or on a workbook as mentioned earlier. For more explanation concerning the uses for Master Pages, consult the *Xtags User Guide*.

Troubleshooting Tips

There will be times when your document just doesn't behave like you expect it to. You could be faced with a scenario where you are working on a file and when you click on the next spread, it closes down the program, or all of the type might disappear. This can be one of those mentally taxing moments when Xtags really can do some magic for you.

I've seen all kinds of things go wrong in files. These seem to have shown up more in Quark 4.1.1 than any newer version of the program. What I do if I have the program closing out on a certain page, I will:

1. Make sure the file is enlarged enough where the spread before or after isn't visible.
2. Go to the page before the offending spread, and click on the last line. Then I go to the spread following the offending spread and click on the top line and select all the text between. I then select *Copy Xtags Text*.
3. Go into BBEdit, create a new file, and Paste.

This is where you really need to know your codes. Upon pasting the file, I hope to discover the error code in the text. It is possible that it is within a floating element. If this is the case, then I will do a *Batch Export* out of Autopage and try to isolate the error this way. If you know the codes well enough, it's very easy to find the error, but if you are still getting to know the codes, an experienced Xtags user is probably your best bet.

Can't Select First Character of Type (32768 Error)

Let's take a look at some text that was exported. In the paged file, I couldn't select the first character in the paragraph. When I exported the paged file I had this starting the paragraph:

```
@CHAP_BM_FIRST:<t--32768><z--32768><f"?">~NOSTYLE~[ITC
Highlander BookItalic%P%100.0]
<t0><z11><f"Berkeley-Medium">@CHAP_BM_FIRST:<t--32768>
<z--32768><f"?">
<t0><z11><f"Berkeley-Medium">@CHAP_BM_FIRST:This has a lot...
```

In order to get this to page correctly, I deleted the <t--32768><z--32768> and the <f"?"> codes and I reimported. I did a *Copy Xtags Text / Paste* combination and the first character was selectable again without the error codes:

```
@CHAP_BM_FIRST:This has a lot...
```

This "32768" error will show up occasionally and it needs to be deleted out. I've seen it once actually make type disappear. I would go to page 15 of the file and suddenly all of the type would be gone to the beginning of the file. It would still be there, but wouldn't be visible on the screen. This code can show up in strange forms. Here's one I found a few months ago where the tag all ran together:

```
<t--32768z--32768b--32768f"?">
```

Error Appearing in the Last Space

Here is another error that sometimes appears in the paged file. I usually find this on the last space of a text box. I have no idea what causes this, but it can trigger major errors with the type and sometimes keeps pages from being printed. The code will appear like:

```
<B><I><O><S></><V><S><U><+><-><t--32768><h--32768>
<z--32768><b--32768>
```

This usually ends up enhancing the style and putting a shadow on the type. This may seem minor, but it has been a major problem in a few books I've had to correct. I do not have the answer on how these errors get into the files to start with?

Stray Colors in the Paged File

There are times when you will run a *Collect For Output* report against a paged document to discover stray colors throughout the file. One way this happens is if the pager is using an XTension like XMLxt to embed hidden XML codes.

40 *Xtags Maximized*

If the pager decides to have their “Normal” style set to “Cyan” and then imports the text file and a few new undefined styles import into the file. Upon import, they will be in the “Cyan” color. The pager then does the “Convert to Hidden Tags” while they are in “Cyan” and the style is still undefined. They then create the styles and apply them. This may seem acceptable, but here is what happens to the file underneath:

```
@t1:<cK>The teacher<cC><A(3,"A1C\#003",\#000\#007</ITAL>)[474]><cK>
```

The hidden tag is still in the “Cyan” color while the rest of the style sheet is in black. If this is a 4-color title, this is not an issue, but in a 1- or 2-color book, this is a major issue. The problem with using “Cyan” is that it cannot be deleted out of the color palette. My suggestion for not having this problem is if you want your normal to reflect a color, make it a duplicate like “Cyan1” and then delete it after fixing all of the newly imported styles.

To fix this in the current file, you would have to either re-import and alter this (which isn’t a possibility if the file is paged), or export each page carefully and fix the file by deleting the <cC> tag. I would go through and on this line I would re-import it using this text:

```
@t1:<cK>The teacher<A(3,"A1C\#003",\#000\#007</ITAL>)[474]><cK>
```

This will solve the color issue. If you imported a lot of styles and converted the tags, a major problem may be on your hands. I’ve seen this happen to a couple pagers and they spent hours correcting the problem. Thankfully Xtags could get in there and correct the problem or it would have been more of a headache to put back into working order.

3 Building Picture and Text Boxes

CHAPTER

When I was first introduced to Xtags almost 10 years ago, the picture and text box tags were the big selling point. I had no idea how much time I had wasted prior to realizing the power that lies in these tags alone. Many automation awakenings happened for me upon learning about Xtags. The direction I took working with software was never the same following this discovery.

Manually building boxes and bringing in art by hand is like taking a trip by horse and buggy. You'll reach the same destination, but it will take you ten times as long. As long as the objective is the same, why not arrive in style and ahead of schedule. That is where the power really lies here.

I could not work without the text and picture box tags, nor would I. There will still be some adjustments that have to be made to the code, but it's minimal compared to the labor-intensive way of manually setting up your boxes and art. One might ask, "Why do people still do this by hand"? This is one of the more troubling questions facing the publishing industry. After years of paging and setup, I will always do the up-front work to reduce as many manual tasks as possible. Enough time is utilized with the pagination process that it's senseless to manually create art and floating elements.

Xtags Fields

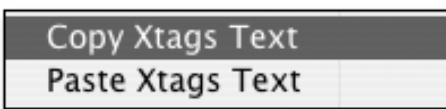
Xtags uses comma-separated fields to process the detailed information for importing boxes and images. The following shows a complete string for importing art:

```
<&pbu2(0,0,46,59,0,0,,n,0,(K,n),(100,100),"Solid",n,0,0,m,100,100,0,0,0,0,"Hard Drive:APManual:Art:ch_001.eps",,,)">
```

These are the fields needed to pull in the referenced piece of art. At first this can be somewhat confusing, but remember that the "&pbu2" is for importing an unanchored picture box, and "&tbu2" is for importing an unanchored text box. There are a multitude of uses for these strings.

With Xtags, a pager or setup person really needs to know what each field represents between the commas. It's very important to be aware of these because you will need to make adjustments.

I've heard many arguments from people that it is important to build your Xtags from scratch all of the time to avoid unneeded content. I don't necessarily disagree that you need to know how to do this, but to get the desired result, you can select the grouping, select the Item tool, and use *Copy Xtags Text* and *Paste* and adjust as necessary.



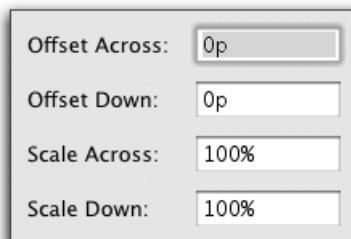
By setting up the preferences correctly, most of the unnecessary content will stay out of the string.

I'll break the fields down quickly. Remember if you need a color in a box, the shortcut colors for any Xtags box are C – Cyan; Y – Yellow; M – Magenta, W – White, and K – Black (which is the default).

Xtags Issues

When using Xtags to bring in photos or boxes, one of the biggest mistakes is ending up with unnecessary information. When using Xtags, there are two ways to get the Xtags strings. One is to type the fields in manually starting with <tbu2(0p, etc.), or by grouping the boxes together, going to *Edit:Copy Xtags Text* and *Paste*. This works, but it is a good idea to do the following:

1. Make sure the runaround is off all boxes, both text and pictures (if a runaround is unnecessary).
2. Make sure, if possible, that the x and y positioning are set as even numbers. It is hard to make sense of an x coordinate at 37p4.997. Try to make it 37p5. It's easier to work this way. This practice should be used with the width and depth of boxes as well.
3. Make sure there is no offset or image scaling. Select *Modify* to fix this or use the bottom right of the palette.



If not, you could end up with mistakes highlighted in this example:

```
<&pbu2(-124,585,46,59,0,0,,i,0,(K,n),(100,100),"Solid",
W,100,(1,1,1,1), m,100,100,2,2.718,0,0,Hard
Drive:APManual:Art:ch_001.eps",,"">
```

The highlighted fields illustrate that the art would always position at $-124 x$, $585 y$, while the “i” (for runaround) is on, affecting the (1,1,1,1) runaround field. The art is offset $2 x$, and $2.718 y$. This would be a disaster if every piece of art came in that way. By checking the piece you’re copying with Xtags, you should end up with more of the result below:

```
<&pbu2(0,0,46,59,0,0,,n,0,(K,n),(100,100),"Solid",n,0,0,m,100,
100,0,0,0,0,"Hard Drive:APManual:Art:ch_001.eps",,"">
```

All of these are set correctly for what is needed. It is essential to start out with the correct amounts in the fields. It is also of great importance to know what each of these fields represents. To better gain some understanding, build some strings from scratch following the Xtags field section in this book, or refer to the *Xtags User Guide* by Em Software, Inc.

Unanchored Picture Box Tags

To try to make this more comprehensible, I’ll break each field down. The screen shots show where this would be implemented within Quark.

1. *x* – This represents the left position (*x* coordinate) on the page of the text box or picture box, typically this should be set at 0 for the top box (unless you have an indent). By using “B” (0 B,0...) after the *x* coordinate before the comma, the art will position on the pasteboard.
2. *y* – This represents the top position (*y* coordinate) of the box where it imports on the page. Normally, this also should be at 0 for the top box.
3. *width* – This is the width of the box. By typing in the maximum width the art can be and then adding a question mark (?) afterward, the box will shrink-to-fit the width of the piece of art.

```
<&pbu2(0,0,15p?,25p,
```

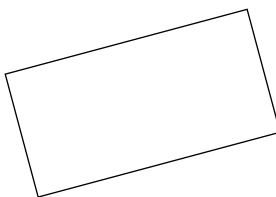
4. *height* – This is the depth (height) of the text box. By typing in the maximum depth of the art can be and adding a question mark (?) afterward, the box will shrink-to-fit to the depth of the art. Make sure the height does not exceed the maximum depth including the pasteboard.

```
<&pbu2(0,0,15p,25p?,
```

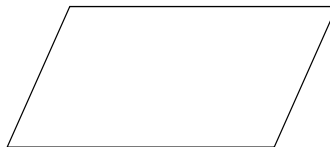
You can also make a box shrink-to-fit with some additional depth by typing a value after the “?” and before the comma.

`<?pbu2(0,0,15p,25p?p4,`

5. *boxangle* – If a box needs to be rotated 45° this is where you’d apply this. You can rotate either direction up to 360°. Here’s an example of how the boxangle will appear with a 15° rotation.



6. *boxskew* – If the box needed to be skewed, you’d do it here in degrees. The range is from –75 to 75°. Here is how the box will be skewed. If placing an image in the skewed box, the image within will be skewed to the same degree of the box. Boxskews can be combined to make interesting effects for backgrounds.


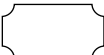



7. *flags* – This is where special handling is applied for the box. If you don’t assign anything here, the box stays to the front. You can apply more than one flag. For example you could have the following acceptable sequence in the tag “lbk”. Selecting “k” instructs the box to be sent to the back. While the *Send to Back* feature is my main use for flags, there are several other options:

b	box print suppression
h	flip contents horizontally
k	send to back
l	lock item
p	picture box print suppression (content)
v	flip contents vertically

8. *runaround* – This is an important field not to leave empty. If you don’t want a runaround on the box you must type an “n”. The default is setup to have a runaround. A box with a runaround will receive an obstruction message during the Autopage pagination process, so unless absolutely necessary, I would not suggest applying the runaround if using Autopage.

9. *frame width* – Type in the needed frame value such as .5 or 1 pt. If a frame is unneeded, leave this blank. You can also place parentheses around this and add corner radius and the corner type for rectangular boxes only. The corner types are as follows:

	v	convex (rounded corners)
	c	concave (inverted)
	s	for straight (corners are angled and straight)

To get the corner radius, you'll need to place twice the amount of the needed corner radius. For example: (1,6p) will give a 1 point frame with a 3p corner radius. Here's an example of a tagged frame:

(.5,2p,s)

For a circle (oval), you will need to use the “o” optional parameter. You cannot have a corner radius on this because this tag doesn't accept it. To get a circle with a 1pt frame, use this code:

 (1,o)

The corner radius and corner type combinations are something you have to work with to get the hang of. I recently created a FileMaker Pro database to use in correlation with Xtags and getting all these combinations to work required a lot of additional scripting.

10. *frame color* – Frame color will default to black; however, if you need a color, type it in the field with the color between quotes: “Cyan1” or “PANTONE DS 10-6 C”. An optional parameter is also included in the color for the *Frame Gap* which would be used on rules like “Double” or “Thin-Thick-Thin”. The first color is for the Frame, the second for the *Frame Gap* as seen here:

(“Color_05”, “Color_01”)

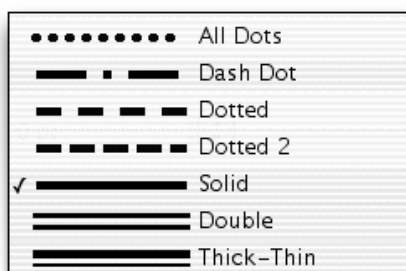
If standard frames are used without the *Frame Gap*, simply putting the color in the field is acceptable. If the color is left off completely, it will default to Black (K)

11. *frame shade* – If the frame needs a shade, the percentage of the frame, 80%, is placed here as “80”, for example. An optional parameter is available for shade similar to the “Frame Color”. This is where the *Frame Gap* is in the next field to be used with frames such as “Thin-Thick-Thin” that can have an additional highlighted color.

Here's an example of this being used where the Frame is 80% and the *Frame Gap* is 50%:

(80,50)

12. *frame style* – If needing a different rule than the standard solid rule, you can apply any of the others by typing in the name from the frame pull down menu. If you need the dotted rule, type in a “All Dots”. The list below shows a portion of the frame names.



13. *bg color* – This is the background color (hue) of the box. It will default to white, but if you want a none background, type in “n”. If color, type in the color such as “Cyan1” or “50M70Y”. Be sure that the color is defined in the Quark document before trying to bring in a box with an undefined color. If *Report errors* is checked, this error will occur:

«Xtags error: No such color: tag &pbu2 param #10»

If *Report errors* is not checked, the import will not produce the picture or text box.

14. *bg shade* – Type in the background shade percentage here. 100% white is the default. Sometimes an image can be placed within the background shade to give a different effect such as:




15. *text outset* – If you have the runaround selected in field #8, enter the runaround amount here. If you need a different runaround on certain sides, the runaround reads (top,left,bottom,right). You can also plug in these amounts for a runaround here, but leave field #8 as “n”. When you turn the runaround on in QuarkXPress 4.1.1, the amounts will hold. Unfortunately, QuarkXPress 6.5 will not hold (at this point). When you turn off the runaround, it defaults back to (1,1,1,1).
16. *placement* – This defaults to manual “m”, but “c” for centered can also be selected. The centered can be very beneficial if you needed all the art to center within the text area. This is achieved by putting in your width at 30p (not shrinking to fit). The example below shows how the art will center in the space within the dotted lines rather than left aligning. This works great when needing the art to center by a width. The white space on the right and left of the image doesn’t cause any production issues.



The other options are “a” and “f”. The “a” option is for expand or shrink-to-fit while maintaining the original aspect ratio. If all of the art was at 30p × 20p, you could use the “a” and the images would reduce at the same percentage to fit to the smallest dimension in the box. In this example, the image maxed out 92% at the width leaving the top and bottom showing the white space:

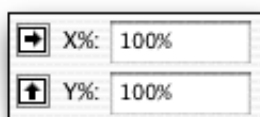


The “f” will expand or shrink-to- fit the box. This is like selecting “ shift f” where it will fill the box with the image. This means the width could be 97% and the depth 48%. Here’s an example of how this would appear. Typically this is used on design elements and not images unless you are going for a unique effect.

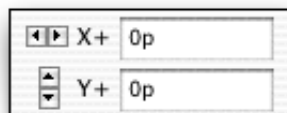


We will cover more about placement in this chapter and throughout the book.

17. *scalex* – The horizontal direction of the art when scaled. Unless you import your art other than 100%, you would rarely want to change this.

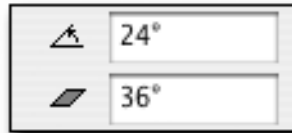


18. *scaley* – The vertical direction of the art when scaled. I rarely alter this field because I generally require my art 100% of the size.
19. *offsetx* – This will put in extra white space from the left of the box. Ideally you’ll want these always set at 0p, unless you have certain circumstances. This can be beneficial if you need a rule added and 12 points white space on the left.



20. *offsety* – This will put in extra white space from the top of the box. Ideally you want these set at 0p, unless your picture has special handling or are trying to trap a rule.

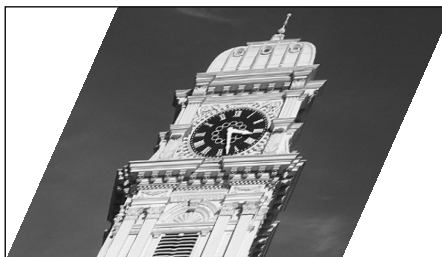
21. *pixangle* – This is the angle in degrees of the image within an unrotated picture box. Do not confuse this with the #5 field. In the measurement palette screen capture below, it's the top field.



Here is how the the *pixangle* will set inside a standard rectangle box. There are many times you may want to use this depending on the effect you are trying to accomplish.



22. *pixskew* – When skewing a picture, this is the amount in the bottom field in the palette image above. The *pixskew* can create some very striking effects if handled correctly.



23. *picture pathname* – This is the path to the art when importing an image. If the translation table is in the same chapter with the art, you only need the art name. The art will automatically find the picture. Otherwise, the entire path from the server or folder will need to be specified similar to: “DualDisk:Xtags:Art:ch01_001.eps”. PDFs are also supported to import.
24. *picture type* – Instructs Quark which type of missing art needs to be updated. The choices are P or p for PICT, T or t for TIFF, and

E or e for EPS. Instead of *Picture Usage* putting in a temporary PICT tag, it will actually specify the image type.

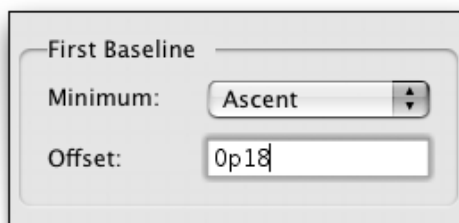
25. *box name* – This is an AppleEvent-relative box name. This should be left blank unless this fits your workflow. If you use AppleScript, this is a way to name the box so the script can find the box.
26. *layer name* – This is for Quark 6.1 and higher. This is for unanchored boxes only. It instructs which layer for the unanchored box to be placed on. If not indicated, it will be on the selected layer. The layer will appear in the Xtags string as shown highlighted below:

```
<&tibu2(0,0,259,137,0,0,,n,0,(K,n),(100,100),"Solid",n,,,m,,,
,,,,"","","New Layer 2")>
```

Unanchored Text Box Tags

The Unanchored Text Box uses the <&tibu2 code. Keep in mind that if only default information is going to be needed, it is acceptable to only include the first four fields. The first 15 Unanchored Text Box fields are exactly the same as the first 15 of the Unanchored Picture Box Tags, so there is no real use in going over these again. We'll start with field #16 and continue to the end.

16. *columns* – By placing a “2” here, this will put in a two-column text box. It's pretty self-explanatory, but the problem is that if you put in a 36p? depth, the first column will fill completely before putting text into the second column.
17. *gutter* – This is the gutter between the columns. By putting 1p in here, it will put a 1p gutter between the two columns.
18. *text inset* – This is where you can set the text inset. If needing 12 points around, put in a 12 here. If you need 12 around all sides, but the bottom needs 9 pts, put in (12,12,9,12). The settings are top, left, bottom, right.
19. *baseline offset* – If an 18 points drop is needed for the text, type in 18 in this field. This affects the *First Baseline* in the *Modify* window:



20. *baseline min.* – I rarely have a need to change this. This is where you direct Xtags to the amount of the *First Baseline*. Ideally, this is set on *Ascent*, which is a blank field, but if you need one of the others, your options are below:

- a Cap + Ascent
- c Cap Height

21. *vert align* – Used if you need a different alignment other than top aligned (which is the default). The options are “b” for bottom aligned, “j” for vertically justified, and “c” for vertically centered.

22. *interparagraph max* – This is only needed if you use vertical justification in field #21. I typically have not needed to use this because the leading and spacing is specified in my workflow.

Be sure to always end any text box with the closing “<&te>” or the text box will continue to pick up unnecessary content and either make Quark close down, or will generate errors. This is probably one of the errors that most coders consistently make.

I wrote an AppleScript that would look through an “<&tbu2” code to make sure a “<&te>” follows it. The code is written to find any “<&tbu2” and any “<&te>” and generates a *Search Result* in BBEdit. The number of occurrences must be equal or one is mismatched. There may be an easier way approach, but this one has proven successful, so I haven’t changed it.

Xtags Caption Positioning

Caption positioning is one of the more critical aspects of the XTension and gives some people the most trouble. When looking at the design, there are generally three ways in which to position the art: above, below, or on the side. For the Autopage workflow it doesn’t matter which side the caption is on because Autopage will place it on either with the *Horizontal Alignment* parameter. If not using Autopage and you are working with a symmetrical design, you will need to base it on averages on which one you think you’ll need the most.

If the caption is below:

<&pbu2 etc. <&tbu2(0 BL1,6,30p,6p?, etc.

The position is 0x bottom left 1, 6 points is the space below the bottom of the art. If on the side:

<&pbu2 etc. <&tbu2(12 TR1,0,10p,10p?, etc.

The 12 TR1 would be the 12 points distance from the top right of the art. The 0 is the 0y position.

Here are some examples of how the different captions and pictures can be handled using the BL1, TL1, and TR1 options.

Side Caption (Top Left)

The top left-side caption starts with the picture box indented 8p and the text box coming in -8p TL1 to accommodate the caption and space between. I only used the first four fields in the text box (-8p TL1,0,7p,6p?,) because there is no special styling in the text box requiring anything other than the defaults for the remainder of the fields.

You can also start with the text box and then have the art TR1, but for demonstration purposes, this made the most sense.

FIGURE 1 Upper
Left Example



```
<&pbu2(8p B,0,280,289,,,n,,(,n),(,100),"Solid",K,0,m,,,,,"Macintosh
HD: Art:wfall.eps",,)><&tbu2(-8p TL1,0,7p,6p?,)>@FIG:<@FIG_NUM>
Figure 1<@$p> Upper Left Example<&te><&g(1,2)>
```

Side Caption (Top Right)

The top right-side caption is accomplished by simply using a 1p TR1 code, which specifies the space between the art and the caption. If using Autopage,

and you are paging a symmetrical design, this only needs to be pulled in one way and the *Horizontal Alignment* parameter can take care of the rest.



FIGURE 2 Upper Right Example

<&pbu2(0 B, 0,280,289,,,n,,(,n),(,100),"Solid",K,0,,m,,,,,,"Macintosh HD: Art:water.tif",,)>&tbu2(12 TR1,0,79,48,)>@FIG:<@FIG_NUM> Figure 2<@\$p> Upper Right Example<&te><&g(1,2)>

Top Caption

The top caption and bottom caption were accomplished by changing the order of the text box and picture box. They are virtually the same other than the order of the captions.

FIGURE 3 Top Caption Example



<&tbu2(0 B, 0,280,48?,,)>@FIG:<@FIG_NUM>Figure 3<@\$p> Top Caption Example<&te><&pbu2(0 BL1,9,280,289,,,n,,(,n),(,100),,K,0,,m, ,,,,,"Macintosh HD:Art:road.eps",,)>&g(1,2)>

Bottom Caption

From working with hundreds of textbooks over the years, it's been my experience that the bottom caption seems to be the way the majority of textbooks are set up. You'll see figures set in a lot of different ways, but on average this is the treatment that is seen most often. As a result, it is the best one to start with in the translation table as a standard.



FIGURE 4 Bottom Caption Example

```
<&pbu2(0 B, 0,280,289,,,n,,(n),(,100),"Solid",K,0,,m,,,,,"Macintosh
HD:Art:city.tif",,)><&tbu2(0 BL1,9,280,48?,.)>@FIG:<@FIG_NUM> Figure
4<@$p> Bottom Caption Example<&te><&g(1,2)>
```

Side Caption (Bottom Left)

The secret to the bottom left caption is thinking it out ahead of time. Obviously, to have the caption bottom align with the figure requires using the bottom vertical alignment “b”. Then I need the figure to read off of the bottom of the figure, so the caption cannot come in first. As you can see from the Xtags string, I have the figure pulling in first at 8p B. That allows me 8 picas to work with from the left edge of the pasteboard for the caption to place. Therefore, I made a –8p BL1 to accommodate the 7p caption and the 1p space to the art.

I allowed 10p for the depth of the caption, so I made the caption box position –9p9, so the baseline of the caption aligns with the bottom of the art, which is a great workaround. If using Autopage, creating a bottom left caption is not necessary if you use the *Horizontal Alignment*, which can bottom align the “Vertical Art Caption.”



FIGURE 5 Bottom
Left-Side Example

<&pbu2(8p B,0,33p?,40p?,,,n,,(n),(,100),"Solid",K,0,,m,,,,,"Macintosh
HD: Art:pwatch.tif",,)>&tbu2(-8p BL1,-9p9,7p,10p,,,n,,(n),(,100),,n,0,,
,,,b,,)>@FIG:<@FIG_NUM>Figure 5<@\$p> Bottom Left-Side
Example<&te><&g(1,2)>

Side Caption (Bottom Right)

The bottom right caption is similar to the top right-side caption in many ways. However, the caption needs to bottom align with the figure, requiring the use of the bottom vertical alignment “b”.

I allowed 10p for the depth of the caption, so I made the caption box position –9p9, so the baseline of the caption aligns with the bottom of the art.



FIGURE 6 Bottom
Right-Side Example

<&pbu2(0p B,0,33p?,40p?,,,n,,(n),(,100),,K,0,,m,,,,,"Macintosh HD:Art:
city1.tif",,)>&tbu2(1p BR1,-9p9,7p,10p,,,n,,(n),(,100),,n,0,,,,,b,,)>
@FIG:<@FIG_NUM>Figure 6<@\$p> Bottom Right Side
Example<&te><&g(1,2)>

Grouping

Following any combination of text or picture boxes, make sure you have the items grouped together using the “&g” code. The full code is <&g(1,2)> for two boxes, <&g(1,2,3)> for three boxes, and so on. The order is based on the recently created boxes or how you want them to be grouped. It is possible to have different groupings in one tag. Here’s an example where there are 2 boxes grouped together, then two other boxes grouped together, then at the end box pairs are grouped:

```
<&pbu....<&g(2,1)><&pbu....<&g(2,1)><&g(4,1)>
```

Anchored Picture and Text Boxes

Anchored picture and text boxes are very similar to unanchored with some differences in the fields. Anchored picture boxes use “&pb” and anchored text boxes use “&tb”. There are many uses from icons to small images. MathType art can also be used as anchored pieces. If you are anchoring type, one advantage is that when exporting the material, the text within the anchored piece becomes part of the text flow.

Be aware that an anchored box cannot contain another anchored box. It’s one of the limitations of QuarkXPress.

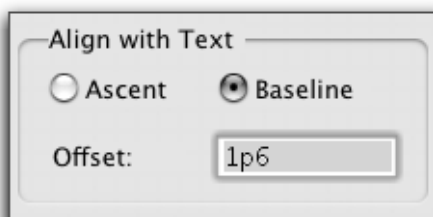
Anchored Picture Boxes

The &pb contains the following fields (parameters):

```
<&pb(width(1), height(2), text align(3), frame width(4), frame
color(5), frame shade(6), frame style(7), bg color(8), bg shade(9), text
outset(10), placement(11), scale x(12), scale y(13), offset x(14), offset
y(15), angle(16), skew(17), picture pathname type(18), picture
type(19), box name(20), layer name(21) )>
```

Most of these have been mentioned in the unanchored section, but I will specify the definitions of those that are different:

3. *Text Align* – When anchoring a piece of art, within the *Modify* window you are given two options. For ascent use “A” or “a”, for baseline use “B” or “b”, which is the default.



If using baseline, the offset also goes within this field in parentheses following the “a” or “b”, separated by a comma. For example:

```
<&tb(5p6,2p6,(b,18)...
```

16. *Angle* – This is the same as *pixangle*, but applies to the anchored box

17. *Skew* – This is the same as *pixskew*, but applies to the anchored box.

Anchored Text Boxes

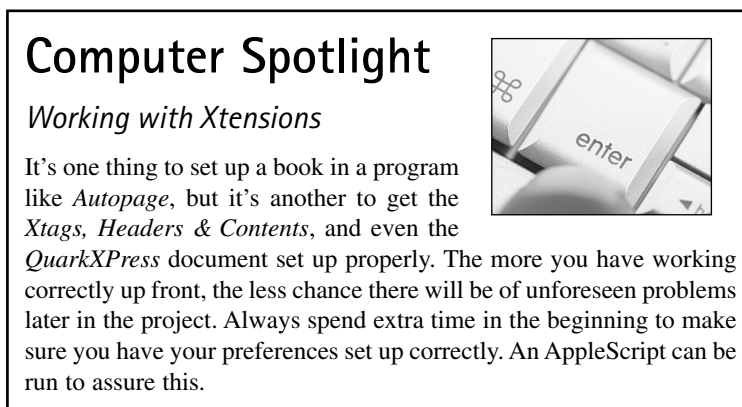
The &tb contains the following fields (parameters):

```
<&tb(width(1), height(2), text align(3), frame width(4), frame
color(5), frame shade(6), frame style(7), bg color(8), bg shade(9),
text outset(10), columns(11), gutter(12), text inset(13), baseline
offset(14), baseline min.(15), vert. align(16), interparagraph
max(17), box name(18) )>
```

These are all covered in the unanchored or in the anchored picture boxes, so we'll move on to some examples.

Anchored Picture Box Example 1

Here's an anchored art example where the anchored art is tabbed to the right where it works as part of the box element. In some boxes, this could also be floating, but it makes it easier to code if the image is always going to be to the right of the title.



The anchored code for this box is as follows:

```
<&pb(82,66,a,0.5,(K,n),(100,100),"Solid",W,100,(0,12,8,8),m,100,
100,-0.14,-0.5,0,0,"56 GB Disk:Xtags Book:Images:Chapter
03:fg03_015.eps",,"")>
```

Anchored Picture Box Example 2

Bulleted lists will occasionally need a graduated bullet in them. This can be done in Quark, but it cannot be done through Xtags because it doesn't support graduated screens at this point. The best method is to make a graduated bullet in Adobe Illustrator and use an anchored picture box. Here's an example:

- This is bulleted list paragraph 1.
- This is bulleted list paragraph 2.

This would require a slight baseline on the anchored picture box. Here is the Xtags code for this including the baseline shift:

```
<b-2><&pb(12,12,,(n),(,100),,,(1,1,1,1),m,,,0,0,,  
"56 GB Disk:Xtags Book:Images:Chapter 03:bullet.eps",,)><b0>
```

Anchored Picture Box Example 3

Pictures will infrequently start off a paragraph sort of like a drop cap will. They begin the paragraph with the text wrapping around the image. I've seen this a lot in cookbooks. They may start the paragraph off with a small image of the food that takes up about 3 lines. Here's an example:



Calamari, Clams, and Shrimp

A new twist for the seafood lover by combining calamari, clams, and shrimp over a white sauce with a few spices thrown in for flavor. Served with fresh garlic bread and a suggested red wine

to compliment the dinner. Read the recipe below:

Usually the picture is in color to give it more commercial appeal. The main details to notice is that there is a .75 rule, the runaround is only to the bottom and the right, and it's an "ascent" anchored box. The anchored picture box code is:

```
<&pb(64,67,a,0.75,(n),(,100),,,(,1,12),m,,,1,,,"56 GB Disk:Xtags  
Book:Images:Chapter 03:fg03_017.eps",,)>
```

Anchored Text Example 1

Here's an anchored box that has a different approach to a design element in this main level head:

Line, Box, and Text Elements

At first look, this element appears to be 3 lines together, but this is actually created as an anchored text box with a 25% fill. There is a style in that text

box called “Line” which has a 3pt 75% shaded rule above and a rule below to make this appearance. Here is the paragraph rule coding that makes this:

```
<*ra(3,"Solid",K,75,0,0,4.8)><*rb(3,"Solid",K,75,0,0,1.75)>
```

The anchored text box has nothing very unusual outside of the 25% shading, and the 1 point runaround. This is coded as:

```
<&tb(21,12.5,,(n),(,100),,K,25,(1,1,1),,,,,,)>@Line: <&te>
```

Anchored Text Example 2

A great use for anchored text is in a situation where you may want to have equations (such as fractions or radicals) that need to be placed throughout the document. In the example shown below, we have a fraction that was created in its own Quark text box requiring two lines to make it up properly. Here’s an example:

The price was $\frac{1}{2}$ the original cost of the software.

The anchored code is as follows:

```
<&tb(4.7,16,b,,(K,n),(100,100),"Solid",W,,(1,1,1,1),1,,,,t,,")>  
@frac:1<b0>  
<k-100>2<k0><b4>--<&te>
```

There are many uses for anchored text beyond math. This can be especially useful in annotated English books where there are text boxes with edit marks that need to be set off. Here’s an example:

1. I can speak almost Spanish after^a_^ few lessons.
2. For three days in Dec^e_^ember, I went to the movies.

To make this work:

1. Create a separate text box that you can put in the annotated mark.

^a_^ or ^e_^

2. In the text flow, just anchor these in where necessary, then kern the lines back. You may have to baseline shift a little, but it’s still better than trying to do this with the actual type area.

in Dec^e_^ember,

3. The Xtags anchored code for this example will look like this:

```
in De<k-70>c<k-80><b-10><&tb(6.45,23.92,b,0,(K,n),
(100,100),"Solid",n,100,(1,1,1,1),1,,,,t,,")>@anno:e
@anno2: ^<&te>@t1:mber,
```

4. With the anchored box put into the translation table, it would be much easier to code for:

```
in De<k-70>c[[anno]]@anno:e
@anno2: ^[[anno2]]mber,
```

5. The translation table would contain these two entries:

```
[[anno]]    <k-80><b-10><&tb(6.45,23.92,b,0,(K,n),
(100,100),"Solid",n,100,(1,1,1,1),1,,,,t,,")>
[[anno2]]   <&te>@t1:
```

Anchored Text Example 3

Another great example of using anchored text is when a dingbat is used in a box to start a section. See the example below



Always spend extra time in the beginning to make sure you have your preferences set up correctly. A script can be run to assure this.

This is an easy box to set up correctly. The main thing is setting up the text box correctly with a “3” Zapf Dingbats in a .75 ruled text box. Here is the anchored text code for this:

```
<&tb(30,30,a,0.75,(n),(,100),,,,(1,1,1,6),,,,24,,,,)>
```

Anchored Text Example 4

Sometimes there will be a head that needs a small disclaimer or additional information alongside of it. Look at this example:

Consumer Spotlight

We will look at many different aspects of consumer goods in this series of examples from leading experts.

This is the type of treatment I’m referring to. I’ve watched pagers who tried to baseline shift these lines to sit like this and their coding was a complete mess. It is much easier to put this in a text box and anchor it following the head. Here’s how the anchored box (including the baseline shift) will be coded.

```
<b-3><&tb(152,15,,(n),(,100),,,,(6,,1),,,,,,>
```

The markup department would code this as follows:

```
@bxtl:Consumer Spotlight [[a1]]@We will look at... [[a2]]
```

We will cover more uses for working with anchored and unanchored text and picture boxes later in the book. Don't be afraid to experiment yourself. Remember to just try things. At first, this may end up taking some extra time, but the new discoveries will more than make up for the loss of production time. I always find that with each finding, so many new doors open up.

Don't Exceed the Page Size of Document

When setting up your picture box tags and captions, be sure that the total depth of both boxes doesn't exceed the available space on the page or pasteboard. I've seen pagers encounter problems importing their images that couldn't get their Xtags to shrink-to-fit correctly. They would have "60p?" for the image and "20p?" for the caption, but they had only 67p total depth on the pasteboard to work with.

It's critical not to exceed the maximum space on the page when setting up. To do this, create a picture box on the pasteboard to see what the maximum depth of is. Once determined, always keep that amount in mind when building boxes and captions. Typically you know what the average depth of the caption needs to be, so don't feel like you have to exceed that just to avoid overflow. I've found most captions below the figure are typically about 9p deep or less. This is a good measurement to go with unless they are side captions, then the depth is not an issue.

Importing on the Pasteboard

In order for the image, box, or text box to import onto the pasteboard, be sure you always change the first 0 to 0 B as follows:

`<&pbu2 (0 B, 0, etc.)`

By putting in "0 B", the art will import on the left pasteboard, so the art is not covering up the pages. This is especially useful for the Autopage workflow where the program will place all of the images, boxes, and other floating elements in their referenced locations. If you would rather work the floating elements all stacked up on the edge of your page, this wouldn't be the ideal workflow to adopt.

Be especially careful when creating the codes for your boxes that if you are not using the "TL1" or "BL1" type of correlation to your previous box that you'll need the "B" in those as well or part of the box will be on the pasteboard, and the remainder will end up on the page giving messy results.

Unanchored Lines

One major Xtags advancement is the unanchored (&lbu) and anchored (&lb) lines feature. I stumbled across this on accident one day because I'm always

62 *Xtags Maximized*

testing different things to see if something new was introduced when I get an Xtags update.

In the past the way to get a line to import with Xtags was to draw a very thin 1pt box and treat it as a line instead. This still works, but is kind of pointless unless the line needs to have relative placement. At this point, I cannot get relative placement to work with the lines which makes me believe it isn't supported yet. If using an unanchored box to make a line, keep in mind that the line will have to be at least 1pt since Quark does not support a box that is smaller than that size.

The unanchored line is set up in this way:

```
<&lbu(176,134,0,212,r,n,6,("Blue","Green"),(100,15),"Thin-
Thick",0,"Plain","2","")>
```

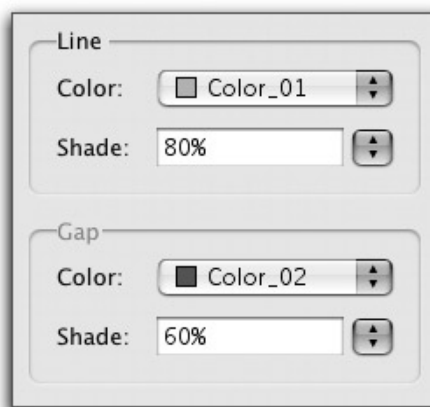
The breakdown for each unanchored line field is as follows:

1. *x* – This represents the left position (*x* coordinate) on the page of the line. By using “B” (0 B,0...) after the *x* coordinate before the comma, the art will position on the pasteboard. You can use the BL1, TL1 positioning on the *x* and *y* during placement.
2. *y* – This represents the top position (*y* coordinate) of the box where it imports on the page. Normally, this also should be at 0 for the top box.
3. *angle* – This is the angle from the terminal point of the line.
4. *width* – This is the width of the line from the terminal point.
5. *flags* – This is where you can have special handling applied for the line, including the type of line it will be. You can apply more than one flag. For example, you could have the following acceptable sequence in the tag “orlb”. Here are the options:

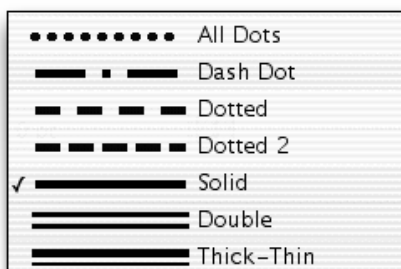
r	line tool (ray)
o	orthogonal tool
b	box suppression
l	lock item

6. *runaround* – The runaround is self-explanatory. It puts a runaround around the line. The amount of the runaround is specified in field #11. The default is “n” or blank, and “i” for runaround similar to the box and text box feature.
7. *width* – This is the width of the line such as 0.5, 1, or 2.
8. *color* – For lines that don't support the “Gap” feature, the color will be in it's own field such as ,“Color_01”, or if there is a “Gap”, then

you would have the line color first and the color gap second in parenthesis such as (“Color_01”, “Color_02”). As mentioned earlier, the color must be defined first. Here’s how the Line and Gap will appear in the *Line Modify* window in QuarkXPress:



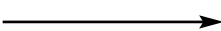
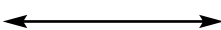




9. *shade* – The shade can be in it’s own field such as ,“60”, or if there is a Gap, then you would have the line shade first and the gap shade second in parenthesis such as ,(“80”, “60”).
10. *line type* – If needing a different line than the standard solid line, you can apply any of the others by typing in the name from the frame pull down menu such as “Dash Dot”, “Dotted”, “Dotted 2”, etc. The list below shows a portion of the frame names.



11. *runaround amount (outset)* – If you have the #6 runaround let for “i”, you will need an actual runaround amount listed in this field. For lines, the runaround can only have a single “Outset” amount. The lines feature does not support top, left, right, or bottom runarounds like picture and text boxes.
12. *arrow type* – This is actually for what type of arrow you will use with your line. In most cases you probably will leave this as “Plain”, but

there may be times when you'll want an arrow at the end of your line. The options are as follows:

Plain	
Left	
Right	
Double	
Left Feathered	
Right Feathered	

25. *line name* – This is an AppleEvent-relative box name. This should be left blank unless this fits your workflow. If you use AppleScript, this is a way to name the line so the script can locate it.
26. *layer name* – This is for Quark 6.1 and higher. This only works for unanchored lines. It instructs which layer for the unanchored box to be placed on. If not indicated, it will be on the selected layer.

Unanchored Line Example

I think this is a good unanchored line example to work with because it shows different treatments for the lines.

Screenwriting Resources

Internet There is an enormous amount of documentation on the Internet concerning the formatting, style, and boundaries of writing screenplays.

Magazines Many online and printed magazines contain all the resources you will need to be up and running.

Libraries Libraries are a fantastic place to locate screenplays. Screenplays such as "Napoleon Dynamite" and "Dream Theater" are available in bookstores.

Here is the code that makes this up. The unanchored lines (&lbu) include BL1 for the closing line.

```
<&tbu2(0,0,27p,35p?p2,,,n,,(K,n),(100,100),"Solid",K,12,,1,,(11,14,9.5,
14),,,t,,,"")>@bx_tl:<B>S<$>screenwriting <B>R<$>esources
@bx_tx1:<B>Internet<$> There... "Dream Theater" are available in book-
stores.<&te><&lbu(0 BL1,0,0,27p,or,n,4,K,35,"Solid",0,"Plain",,"")>
<&lbu(8,0,270,35.85,or,n,4,K,35,"Solid",0,"Plain",,"")><&lbu(6,34,
0,318,or,n,4,K,35,"Solid",0,"Plain",,"")><&g(4,3,2,1)>
```


1. The first line following the end of text “<&te> is the bottom line. Since the positioning depends on the depth of the text box, this required a “0 BL1, 0” code on the *x* and *y* coordinates so the line would split the gray box and trap correctly.
2. The second line is the horizontal line extending underneath the title.
3. The third rule is the vertical line that extends down from the top of the box meeting with the second horizontal line. Neither the second, nor the third line need the BL1 or TL1 treatment because they will always be in this position.

Anchored Lines

Anchored lines are very similar to the unanchored lines with a few differences that will be mentioned. Here's an example of how an anchored line appears:

```
<&lb(0,82,or,b,l,K,100,"Solid",1,"Right Feathered",")">
```

The breakdown for each unanchored line field is as follows:

1. *px* – Horizontal. This is the line's initial point to the terminal point. Basically, if it's positive, it points the right side, if it's negative, it points to the left. If the “r” flag is selected, this becomes the polar angle.
2. *py* – Vertical. A positive value moves towards the bottom of the page.
3. *flags* – Same as Unanchored Lines.
4. *anchored alignment* – When anchoring a line, within the *Modify* window you are given two options. For ascent use “A” or “a”, for baseline use “B” or “b”, which is the default.
5. *width* – Same as Unanchored Lines.
6. *color* – Same as Unanchored Lines.
7. *shade* – Same as Unanchored Lines.
8. *runaround amount (outset)* – Same as Unanchored Lines.
9. *arrow type* – Same as Unanchored Lines.
10. *line name* – Same as Unanchored Lines.

Anchored Line Example 1

Instead of using an underscore or a tab to make a line within the text for an answer or write-on line, I prefer to anchor the line in the text flow. The problem with just using the tab or a underscore depends greatly on the *Kerning Table's* tracking on that particular font. By using a line, the guarantee is there

that the lines will not be broken up and will be solid. Below is an example of a line that is anchored in the text flow.

What was the price of the software \$_____?

The code for the anchored line is:

```
<&lb(0,64.919,or,b,0.5,K,100,"Solid",1,"Plain","")>
```

Anchored Line Example 2

Here is another anchored line example. This is actually something I've seen quite often in college textbooks. It's when you have an example or a vignette that has a vertical line running the length of it. This would be great for an anchored line. Here's the way it appears:

EXAMPLE 2

Hollywood has always had it's share of celebrities who become famous merely because of the success of one movie. Julia Roberts had been roughing it in movies like *Satisfaction*, *Mystic Pizza*, but once she landed the role in *Pretty Woman*, suddenly she was the a-list actress.

The next two years saw her appearing in one film after another, most without leaving any lasting impression. It wasn't until she backed off for a couple of years before she found returned box office success in movies like *Notting Hill* and the academy award winning *Erin Brockovich*.

By looking at the example we see a vertical line and a small horizontal rule where the example ends. The small horizontal rule is done in the style sheet on the last style for the example. The vertical line is an anchored line with “ascent” selected and a baseline shift of “2”. The length of the line can be coded at any length and just pulled down to connect to the closing horizontal rule. Here is the code for this.

```
<b2><&lb(270,125.75,or,a,1,K,100,"Solid",1,"Plain","")>
```

We'll have more examples of anchored and unanchored lines in upcoming chapters. This should give you a good example of some of the uses that can be applied using the lines feature.

4

CHAPTER

Image and Caption Building

Working with figures and boxes can really make or break the profitability of a project depending on the time you dedicate to this prior to starting the first chapter. Many users of Autopage have the tendency to build a shell for the box and put it in the library using the T=E method. This is an acceptable way of working with boxes, but there is still adjustment. Autopage recently made a box feature that will page the boxes, but there are still some limitations with this, although it's very impressive. Not every reader uses Autopage, so this will not apply to everyone.

For users of Xtags who don't use Autopage, boxes do not have to be so complicated that you need to do a lot of hand manipulation afterwards. So much can be achieved using the shrink-to-fit, relative placement, and BL1 type of placement options. The same with figures. When figures just have an image and the caption, the code is relatively easy to deal with. However, you're only scratching the surface on what is possible. The last chapter we covered the anchored and unanchored picture, text, and line usage. This is the foundation of all boxes and images.

This chapter is going to break into expanding what you know about working with figures and getting away from the hand manipulation that you might be used to doing. We'll start by covering a little more about some of the options available with the unanchored and anchored tags.

Shrink to Fit Capabilities

In the last chapter, it was mentioned a little bit about the shrink-to-fit capabilities of the unanchored and anchored picture and text box tags. This is something that gives power to what you are trying to achieve. By putting in a "?" following the width or depth, the box will shrink up to the content. Because of this, you will usually not have to do any hand manipulation afterwards. The only time this does not work is on the width of the text box because Xtags does not know what width it would need to shrink to. By using

relative placement, there are possibilities here, but we'll cover that in the next section. To shrink to fit a box, the code will reflect this:

```
<&pbu2(0 B,0,29p?,48p?,
```

There will be times when you want to cheat this a little. For example, you may want the figure to have a .5 rule around it and want the image to trap within the rule to avoid any white lines between the image and the rule. In this case you would need the shrink to fit to reduce the size by –1 point. To do this you would subtract 1 point from the width and depth of the image, but have the *offset x* and *offset y* be “-.5” as shown below:

```
<&pbu2(0 B,0,30p?-p1,40p?-p1,,,,0.5,(,n),(,100),,,,(1,1,1,1),m,,,
-1,-1,,, "56 GB Disk:Xtags:Images:Chapter 06:fg06_010.eps",,,)>
```

Adding additional space to an unanchored box can be necessary if you don't want to use the insets and have a *First Baseline* instead. At the bottom, you'll do a shrink-to-fit, but will need to add extra space so there is a proper inset all around the text. The code would look like this:

```
<&tbu2(0 B,0,30p,40p?p4.5
```

This will add that extra spacing that is necessary. It is good to play with this feature to see what the best workflow will be.

Relative Caption Placement

Relative placement is one of the defining options in Xtags picture and caption placement that gives the program greater control over the art. Relative placement will shrink-to-fit the caption to the width of the art for a horizontal photo. For a vertical photo, it has the capability to shrink-to-fit the caption's depth to the depth of the photo. This is invaluable because it reduces the need to manually have to adjust the art. The coding for this is as follows:

```
(30p,R,0,1")
```

This means:

- 1st field** – The caption will shrink-to-fit starting at 30p (or whatever value you place here), and shrink down to any art 0p1 to 30p.
- 2nd field** – R for Relative placement.
- 3rd field** – This is the amount the caption can be reduced or expanded from the right of the art (on horizontal captions) or the bottom of the art (on vertical captions).
- 4th field** – This is the minimum inch width or depth (depending on if horizontal or vertical) the caption must be. In this case, I'm putting in 1 inch.

For this example, I put a gray screen on the box to show the length of the caption against the image above it.



This is the proper placement for a relative horizontal caption placement

< &pbu2(0 B,0,28p?,48p4?,,,n,,(K,W),(100,100),,K,0,,c,,,,,,,"56 GB
Disk:Art:waterfall.eps" ,,")>< &tbu2(0 BL1,10.5,(28p,R,0,1"),3p?p2,
,,,n,,(,), (,100),,K,15,,,,,,c,,,")>@Label:This is the proper placement
for a relative horizontal caption placement < &te>< &g(2,1)>

Indented Relative Placement

There might be an example where you'll need the caption to be indented 1p from the left, and then you'll need to have the right side pull in 1 pica.



The indented relative caption

In this instance the code would look like this.

```
<&tbu2(12 BL1,10.5,(28p,R,-2p,1") ...
```

The reason I need -2p in the 3rd field is that I've already pulled the text over p12 BL1 when the caption started.

Vertical Relative Placement

Vertical Relative Placement works great when you have a photo that needs the caption next to it to be the same depth. This usually happens with shading or with a rule around the caption. The positioning on this is used with the TL1 or TR1 as (46p4,R,0,1"). Here is how this will import:

This is the caption
for a relative
vertical placement
in Xtags. A great
use for this is a
design with a
shaded box.



Here is the Xtags string for the above image and relative caption:

```
<&pbu2(14p B,0,30p?,46p4?,,,n,,(K,W),(100,100),,K,0,,m,,,,,  
,,,"56GB Disk:AP_Book:Art:beach.tif",,"")><&tbu2(-13p TL1,0, 9p,  
(46p4,R,0,1"), ,,,n,,(K,W),(100,100),,K,10,,,9,,,c,,,)>@Label1:This  
is the caption for a relative vertical placement in Xtags. A great use  
for this is a design with a shaded box.<&te><&g(1,2)>
```

Vertical and Horizontal Mixed Relative Placement

The relative placement can also be used both vertically and horizontally at the same time. This is extremely useful when you have a piece of art that may have a decorative border or some other type of attached art element needing to be ignored during placement. Autopage has a C=O option in the art, tables, and side art that will position only the information in the text box containing the C=O tag. This can be placed over the top of the photo to ensure this type

of placement. I mention Autopage a lot because I am a power user of Autopage and sometimes I think some pagers miss the connection that if they use Xtags to gain productivity that they should also look at Autopage to gain even more efficiencies. Here is an example with decorative circles around the figure for this type of placement as well as an Autopage tag within:



To achieve this, follow these steps:

1. Start out by getting the code for the main image. Either write the string or do an *Edit:Copy Xtags Text*. At first you'll only receive the string for the image, which, after adjustments, will be something like:

```
<&pbu2(2p9,3p3,30p?,30p?,,,,n,,(n),(,100)),K,0,,m,,,,,"56 GB  
Disk:Book:Photos:flowers3.tif",,,)>
```

2. The relative placement needs to be both the width and the height. Therefore, you'll need to start top left of the picture box with the text box over the photo. I use the relative maximum column width (**30p4,R,0,1"**) and maximum depth (**50p4,R,0p,1"**). This code will need to include the **<&te>** to close off the text box.

```
<&tbu2(0p TL1,0,(30p4,R,0,1"), (50p4,R,0p,1"),,,,n,,(,),(,  
,n,)@FIG:[[SR A C=O]]<&te>
```

3. The circles are a little more of a challenge because of their positioning. Xtags has the ability to include corner radius by using parentheses in the frame width field, separated by a comma. In this case, the frame width is 1 point followed by a comma; then place two times the amount of the

needed corner radius. This results in a 6p width; therefore this field will have (1,12p). The circle has a 25% black rule and the circle needs to be sent to the back (k). The code is:

$$\langle \text{p}^2(0,0,9p_5,9p_5,\dots,k,n,(1,12p),(n),(25,100),K,0,m,\dots,"",\dots) \rangle$$

4. This code in number 3 will be used twice with a small variance. The top circle positions at 0x, 0y. The second circle will be positioned based on the ending depth and width of the photo. Therefore, it will need a -6p9 BR2, -6p9. Since it also positions under the photo, it is sent to the back.

The final code when all put together will read as follows:

<&pbu2(2p9,3p3,30p?,30p?,,,n,,(,n),(,100),K,0,,m,,,,,"56 GB
Disk:Book:Photos:flowers3.tif",,) ><&tbu2(0p TL1,0,(30p4,R,0,1"),
(50p4,R,0,p,1"),,,n,,(,),,(,),n,>@FIG:[[SR A C=O]]<&te>
<&pbu2(0,0, 113,113,,,k,n,(1,144,),(,n),(25,100),K,0,,m,,,
,,,,"",,) ><&pbu2(-6p9 BR2,-6p9,113,113,,,k,n,(1,144,),(,n),
(25,100),K,0,,m,,,,,"",,) ><&g(1,2,3,4)>

By using the C=O in Autopage, only the relative area will position in the column when paged. The actual placement of the photo with the decorative circles will appear placed like this example:

In Maine, the ocean temperature rarely exceeds 65 degrees in the hottest parts of the summer. There is approximately one half a cup of salt per gallon of water on the nearby beaches. Sand Beach is nestled between mountains and rocky shores to the east of Mount Desert Island.

Megadunes were first noticed by pilots flying over parts of East Antarctica. Satellites showed that megadunes areas were huge areas. The found one area to be the size of California. Nearly 25 percent of the Northern Hemisphere is covered by permafrost from freezing and thawing temperatures. Seasonal snow cover covers over 30% of the Earth's total land surface. Over 95% of the total seasonal snow cover is located in the Northern Hemisphere. Snowfall accounts for over 70% of annual precipitation in the Rocky Mountains and other mountain areas.


Black sand beaches are very common in Hawaii. There's Moana Lei and Moana Kea and there's Hana which is part of the Maui landscape, but a treacherous road will take you to it. Not for the faint hearted, but worth the drive. The scenery along the way is nothing short of breathtaking.

There's also Punalu'u Black Sand Beach in Hawaii. Among other beaches are Lapakahi State Historical Park, Mahukona Beach which used to be an abandoned harbor once used by an Hawaiian sugar company.

Makalawena Beach contains a rocky area and sandy beaches while Onekahakaha Beach Park features one of the most breathtaking views of mountains in the distance.

A small walk down an rocky cliff leads to Papakole Green Sand Beach. For very private times, Pine Trees

Beach is a favorite for travelers who are looking for tranquility over mass cultural appeal. Spencer Beach Park is protected by a giant reef. Other beaches worth investigating are Puako Bay, Pololu Valley Beach, Manini Beach, Kona Cost State Park and Kua Bay. All free of commercialism that surrounds.

A close-up photograph of a bouquet of dark, possibly black or very dark red, roses. The roses are arranged in a cluster, with some fully bloomed and others as buds. Interspersed among the roses are small, delicate white flowers and green foliage. The bouquet is tied with white ribbons that have a subtle pattern. The background is a soft, out-of-focus light color, suggesting an outdoor setting.

Without the assistance of Xtags in the building of this box, this process could be very time consuming.

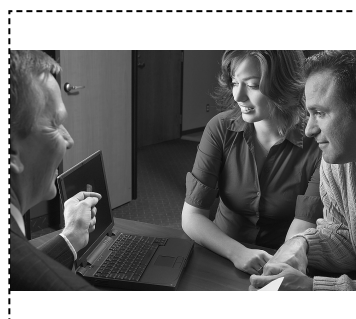
I highly recommend experimenting with relative placement when you have some spare time. The best way to learn how to use this isn't necessarily when you are under a difficult deadline, but rather by testing. When I became aware of this feature, I tried vertical captions, horizontal captions, shortening and lengthening captions, and so on. Some can work under the pressure of a

deadline with positive results, but I find many people get frustrated and return to the manual way of doing things.

Fit-to-Height, Fit-to-Width

The *Fit-to-Height*, *Fit-to-Width* (f) feature is one that works great when you have a certain amount of space allotted for an image and the size of the image hasn't been determined. You must pick either the width or the height for this. Both cannot be chosen simultaneously.

This is not the same as using the using the picture #16 placement option that supports the shrink-to fit by using the “a” or “f” option. The “a” option will make both 80.4%, but would have white space above and below the image in the box. The “f”, in the #16 field, would scale the width to 80.4%, but leave the depth at 100%, thereby distorting the image. Below are examples:



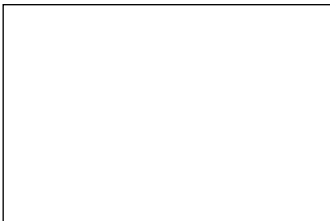
The “a” placement option



The scaled “f” placement option

Don't get thrown off that the “f” is also used for this option. It operates out of field #3 or #4 (#1 and #2 for anchored), so it should be less confusing.

For example, let's say we are working with a box as shown here:

Computer Spotlight	
<p>This book has not covered every aspect because that is not my intention. My main objective is to show the user how to get more out of the program by approaching the program in a different manner than the user might be used to.</p> <p>After following through the book and learning how to do all of the procedures, Chapter 9 is full of step-by-step examples that will give you an opportunity to apply the various functions of the program and it's many uses.</p>	<p>I feel intermediate and advanced users need to work through these examples.</p> 

We can see that an image needs to be placed. We know the width needs to be 18p due to the specifications. Maybe the art department has this at 24p because it is from a previous book or a custom publishing job. What we will

74 Xtags Maximized

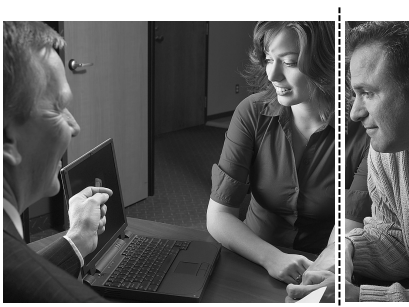
need to do is use the *Fit to Width* feature and let the height shrink to fit as usual. The *Fit to Width* or *Fit to Height* feature is using the “f” inside the width or height tag in parenthesis similar to how relative placement works.

To do this:

1. Write the Xtags code for the box placement as you normally would. Normally it will be:

```
<&pbu2(0 B,0,30p,45p?,,,n,,(n),(,100),,,,c,100,100,0,,,
"56 GB Disk:Xtags Book:Images:Chapter 04:fg04_006.eps" ,,,)>
```

2. In this case, we know we'll need the width to be 18p. We could just put in 18p for the width, but the picture could get cut off as shown below. The dotted rule shows the amount of image that would be cut



We want to avoid this. So, instead, we'll put in (18p,f) for the width. This is telling Xtags that the width is 18p, we want to fit the image to that size, but still have the depth shrink-to-fit.

```
<&pbu2(0 B,0,(18p,f),45p?,
```

3. When the image comes in, it now fits the width perfectly by scaling it to 80.4% both for the depth and the width. It used the width to do the fit, but did a shrink-to-fit on the depth and maintained the 80.4% upon importing the art.

Computer Spotlight

This book has not covered every aspect because that is not my intention. My main objective is to show the user how to get more out of the program by approaching the program in a different manner than the user might be used to.

After following through the book and learning how to do all of the procedures, Chapter 9 is full of step-by-step examples that will give you an opportunity to apply the various functions of the program and it's many uses.

I feel intermediate and advanced users need to work through these examples.



So this is a very powerful feature and, if used properly, will open the door to new possibilities. This function works with both unanchored and anchored picture boxes. I would suggest working with this feature and trying different scenarios to understand where it will help you later.


Forced Anchored Leading

Forced Leading is another optional parameter that works with only anchored text and picture boxes. How this works is by using the (upper or lowercase) “L” parameter within the height tag. Similar to how this worked with the *Fit-to-Height* option. The code would be written like this:

```
<expb((4p,f),(2p6,L) ...
```

This automatically adjusts the leading on the line containing the anchored object. The program *PowerMath* appeared to give this type of result with the equations. Here’s an example (using baseline anchoring) of what would happen if you anchored the image without using the “L” option.


My main objective is to show the user how to get more out of the program by approaching the program in a different manner than the user might be used to. work with the examples and see if you can generate the same results available resources.



This book has not covered every aspect of this program.

The image covers up the type above, which is unacceptable. Even with a runaround, it still doesn’t adjust the type above. However, by putting in the “L” option, we get a different result entirely:

My main objective is to show the user how to get more out of the program by approaching the program in a different manner than the user might be used to. It is best to work with the examples and see if you can generate the same results with the available resources.



This book has not covered every aspect of this program.

By using the “L” option, the leading is now set at 33 pt for that paragraph. It finds the proper amount of leading to accommodate the image. This would pose as a problem if the anchored image fell on a paragraph with multiple lines. The entire paragraph would spread out to the 33 pt.

There are a lot of uses for this. Anchoring *MathType* figures is one that instantly comes to mind. Other uses could be for images that fall directly after the paragraph that may be centered in the text area rather than having to manually place them.

Additional Height Options

Similar to the relative placement seen earlier in this chapter, Xtags offers some additional parameters in the height options that you can use in tricky placement situations. Take a look at this tag below:

(13p, SF, p2,10p)

This contains the different options in this tag that would go in field #4 (unanchored) or field #2 (anchored). This is broken down as:

(height, sizing options, size adjustment, minimum height)

The *sizing options* can be the R (relative), F (fit-to-box-width), S (shrink-to-fit), or L (forced leading—anchored only) options. A combination of these can be added together as shown in the above example. The *size adjustment* is telling the box the amount of additional depth can be added. This sometimes helps when creating boxes. The *minimum height* is the smallest depth allowable. For anchored images, an additional parameter is available at the end for additional leading. The tag would look like:

(13p, SF, p2,10p,5pt)

Working with Figures

In Chapter 3, I displayed examples of different ways that figures can work with caption placement. Those examples were constructed of only a picture box grouped with a text box. I have found figures and captions are not always so basic. At times, the designer will go out of their way to make the figure/caption combination a little more difficult to work with by trying to give it more appeal.

I'm going to show how you could go about making different figure treatments. Some might be self-explanatory, but it has been my experience that a lot of pagers see images/caption groupings like this and just panic. I've been asked many times to set up the Xtags for someone else when the slightest variance has been made in the figure/caption combinations.

If you happen to already know how to achieve some of the examples, don't feel like you need to change your approach. I have found that there is always more than one way to do things, but sometimes it's worth investigating how another person is achieving the results. I have found that I have saved much time in the past just seeing another's approach. It sometimes opened the door to other options I hadn't thought of.

Space Between Rule and Figure

I've seen a great number of designs needing 12 points white space between the figure and the rule around it. Some pagers may feel they need the art department to change the images to include the rule and the space, but Xtags

can handle this quite well. For demonstration purposes I've shown a photo, but typically this is used for line art.



1. The figure will need to have an x and y *offset* of 12 and 12.
2. Add 12 points to the height and the width following the shrink-to-fit. There is actually 24 points extra needed, but you take half of that amount.

```
<pbu2(0 B,0,35p?1p,50p?1p,,,,n,0.5,(n),(100),"Solid",K,0,,m,
,,12,12,,,"56 GB Disk:Xtags Book:Images:Chapter 04:fg04_008.eps",,)>
```

3. By doing this, all the images will import with this space.

Rule Around Picture and Caption

I have actually witnessed a pager set up the figure and the caption with Xtags, but left the surrounding border off. I asked them why they refrained from adding the code for the border and they said, “Well, it can’t be done with Xtags.” I’m here to say that this can. Here’s an example of how it appears:



Figure 2-3
*It is very important
in photography to
pay careful attention
to the small details.*

This all comes down to knowing how to properly use relative placement. I'm going to show you two ways to achieve this. The first is with relative placement. This would pose a problem if you were using Autopage and this was a symmetrical design. In this case, you would need to have the caption positioned on the outside of the page. Anytime you have a grouping of 3 boxes, this won't work correctly in Autopage for that feature.

Example 1: Using Relative Placement

For the example at the bottom of the previous page to import correctly, follow these steps:

1. The maximum figure width and depth is 31p × 45p.
2. Knowing this maximum width and depth information, we'll want the image to begin with a 10pt inset from the top and left of the pasteboard:

`<&pbu2(10 B,10,31p?,45p?`

3. Determine the caption size, which for this example is 6p6 with 9pts between the art and the caption. We will need to use 9 TR1:

`<&tbu2(p9 TR1,0,6p6,15p?`

4. The rule will be offset 9pts around the image. This will need to have a -p9 TL1 position with -9 at the top. By using relative placement for the width, we must add the space of the caption, space between caption, and the 9 pts on the left side of the figure and the right side of the caption. This is (6p6 + p9 + p9 + p9 = 8p9). We'll also need 1p6 for the relative placement for the white space above and below the image. We then have these two relative tags:

`(30p,R,8p9,6p) and (40p,R,1p6,6p)`

5. By putting all of this together, we end up with the following tags:

```
<&pbu2(10 B,10,31p?,45p?,,,,,,(n),(,100),,,,,m,,,,,"56 GB Disk:
Xtags Book:Images:Chapter 04:fg04_010.eps",,,)>
<&tbu2(9 TR1,0, 6p6,15p?)>@fgn:Figure 2-3
@fgc:It is very important in photography to pay careful attention to the
small details.<&te><&pbu2(-9 TL2,-9,(30p,R,8p9,6p),(40p,R,1p6,6p),
,,k,,.5,(n),(,100),,,,,m,,,,,"",,,)><&g(3,2,1)>
```

This is one method of doing this. It's very effective and is a good way to learn the relative placement options and taking them a step further.

Example 2: Using the Rules to Offset

The second example is a little less involved and is actually easier with one less grouping. It's very similar to the *Space Between Rule and Figure* example previously shown. What we'll want to do here is to bring in the image 9pts from

the top and left of the rule and extend the border past the caption. This can be tricky, but following the steps will get this to work. To do this:

1. The maximum figure width is $31p \times 45p$.
2. Since the rule will be part of the picture box, we'll need to have a 9pt offset on the box and have the additional space built into the width and depth. To accommodate the space for the caption, we need to add 9pts of additional space to the width. I've found that this is 50% of the additional width or depth needed. Don't add in the inset because that doesn't affect the additional spacing. Here is the opening picture box tag:

```
<&pbu2(0 B,0,31p?4p4.5,45p?p9,,,,,0.5,(n),(,100),,,,,m,,,9,9,,,"56 GB
Disk:Xtags Book:Images:Chapter 04:fg04_010.eps" ,,,)>
```

3. The text box “-7p3.5 TR1” reads off the right corner of the surrounding border, and includes the caption width, 9 pts for the inset, and .5 pt for the border.

```
<&tbu2(-7p3.5 TR1,9,6p6,21p?,,,,,(n),(,100),,,,,,,>
```

4. The complete tag will look like this with only 2 boxes necessary:

```
<&pbu2(0 B,0,31p?+4p4.5,45p?+p9,,,,,0.5,(n),(,100),,,,,m,,,9,9,,,"56
GB Disk:Xtags Book:Images:Chapter 04:fg04_010.eps" ,,,)><&tbu2(-
7p3.5 TR1,9,78,21p?)>@fgc:Figure 2-3
@fgn:It is very important in photography to pay careful attention to the
small details.<&te><&g(2,1)>
```

Whether you choose *Example 1* or 2, you will get the same result. *Example 2* might be a less complicated approach because it has the least amount of steps.

Labels Below Art (Alphas)

Collectively, pagers have never been too thrilled about putting alpha labels under art. It's my opinion that the art will not be compromised as long as the original is treated as a support piece. This opinion doesn't change the fact that pagers still have to complete this manual and tedious process. There are ways to reduce the time spent on this issue.

In the translation table, it's easy to have a piece of art import in and by just adding something simple like an `[[a]]`, `[[b]]`, or `[[c]]` code, you can have a quick solution to this task. One thing to keep in mind is that on your `<&tbu2` code that if only default information is going to be in the code, it is acceptable to only include the first four fields.

I achieved this by putting these ahead of time into my translation table:

```
[[a]] <&tbu2(0 BL1,p6,(41p3,R,0,1"),p12)>@Label:<*C>(a)<&te>
[[b]] <&tbu2(0 BL1,p6,(41p3,R,0,1"),p12)>@Label:<*C>(b)<&te>
[[c]] <&tbu2(0 BL1,p6,(41p3,R,0,1"),p12)>@Label:<*C>(c)<&te>
```

The major difference here is that on the last label, you will need to add the caption to this. I typically group these together separately with each piece of art. If you attempt to have all the pieces pull in together using the shrink-to-fit, the depth could eventually become too long and an Xtags error will appear. On the following page, I do show how to pull them in side by side.

To keep track, I may even make the label display the chapter number. For example: "@Label:<*C>24(a)". I will delete the "24" once I group the three parts of the figure together. The alphas combined with the art and caption will look like:

```
<&pbu2(0 B,0,36p?,48p?,,,,,5,(K,W),(100,100),"Solid",K,0,,c,,,,,"Figure 01_01.eps",",")><&tbu2(0 BL1,p6,(36p,R,0,1"),p12)>@Label:(a)<&te>
<&tbu2(0 BL1,10.5,(36p,R,0,1"),6p6?,,,n,,(K,W),(100,100),"Solid",n,,1,,,,,t,,")>This is the caption below the alpha<&te><&g(3,2,1)>
```

Here's an example of how this will import.



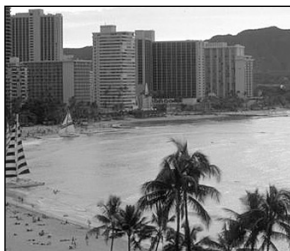
(a)

This is the caption sitting below the alpha.

Bringing in Multi-Piece Figures Together

Until art departments make multiple-piece figures as one piece with support files, there is no getting around multi-piece images. Having multi-piece images doesn't stop Xtags from working. This can fail if you are trying to bring in stacked images because the depth will usually hit the page-depth maximum unless you put a small amount in the total depth of the image. On images that are side by side, this works very well. I find that it is extremely difficult to base align the alphas on multi-depth images because the images will not shrink to fit in reverse. If the alphas should base aligned after importing, *Ungroup* and choose *Item:Space/Align*. The images in the example are the same depth.

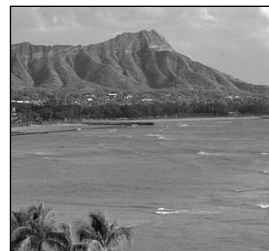
This example contains many Xtags features including TR1, BL1, relative placement, rule trapping, grouping alphas, and so on.



(a)



(b)



(c)

Hawaii's beaches and beautiful scenery are favorites amongst tourists.

1. I first calculate the first image, and then I make sure this is grouped together with the alpha below. For example:

```
<&pbu2(0 B,0,40p?-1,40p?-1,,,n,0.5,(n),(,100),"Solid",K,0,,m,,
,-0.5,-0.5,,, "56 GB Disk:Xtags:Art:Photos:ha1.tif" ,,)><&tbu2(0
BL1,10.5, (35p,R,0,0),2p?)>@ALPHA:(a)<&te><&g(1,2)>
```

2. Next I will need to repeat this almost exactly for the second image, but I'll have this 18 points TR1. The Xtags string is repeated exactly for the third image.

```
<&pbu2(18 TR1,0,40p?-1,40p?-1,,,n,0.5,(n),(,100),"Solid",K,0,
,m,,, -0.5,-0.5,,, "56 GB Disk:Xtags:Photos:ha2.tif" ,,)><&tbu2(0 BL1,
8,(35p,R,0,0),2p?)>@ALPHA:(b)<&te><&g(1,2)>
```

3. The figure caption will need to be added. This will have "0 BL7" so it reads off the bottom of the first image.

```
<&tbu2(0 BL7,9.5,30p,2p?)>@FIG_CAP:Hawaii's beaches and
beautiful scenery are favorites amongst tourists.<&te>
```

Altogether this will need to be brought together with a grouping that will read "<&g(1,2,5,8)>". Here's the full code:

```
<&pbu2(0 B,0,40p?-1,40p?-1,,,n,0.5,(n),(,100),"Solid",K,0,,m,,,
-0.5,-0.5,,, "56 GB Disk:APBook:Art:Photos:ha1.tif" ,,)><&tbu2(0 BL1,
8,(35p,R,0,0),2p?)>@ALPHA:(a)<&te><&g(1,2)><&pbu2(18
TR1,0,40p?-1,40p?-1,,,n,0.5,(n),(,100),"Solid",K,0,,m,,, -0.5,-0.5,
,, "56 GB Disk:Xtags:Art:Photos:ha2.tif" ,,)><&tbu2(0 BL1,8,
(35p,R,0,0),2p?)>@ALPHA:(b)<&te><&g(1,2)><&pbu2(18 TR1,
0,40p?-1,40p?-1,,,n,0.5,(n),(,100),"Solid",K,0,,m,,, -0.5,-0.5,,, "56
GB Disk:APBook:Art:Photos:hawaii3.tif" ,,)><&tbu2(0 BL1,
8,(35p,R,0,0), 2p?)>@ALPHA:(c)<&te><&g(1,2)><&tbu2(0 BL7,
9.5,30p,2p?)>@FIG_CAP:Hawaii's beaches and beautiful scenery
are favorites amongst tourists.<&te><&g(1,2,5,8)>
```

The coded file with macros would be reduced to this:

```
[[f1]]hawaii1.tif[[f2]]@ALPHA:(a)[[f3]]hawaii2.tif[[f4]]
@ALPHA:(a)[[f5]]hawaii1.tif[[f6]]@ALPHA:(a)
[[f7]]@FIG_CAP:Hawaii's beaches and beautiful scenery are
favorites amongst tourists. [[f8]]
```

This truly demonstrates the cost-effective power of the translation table.

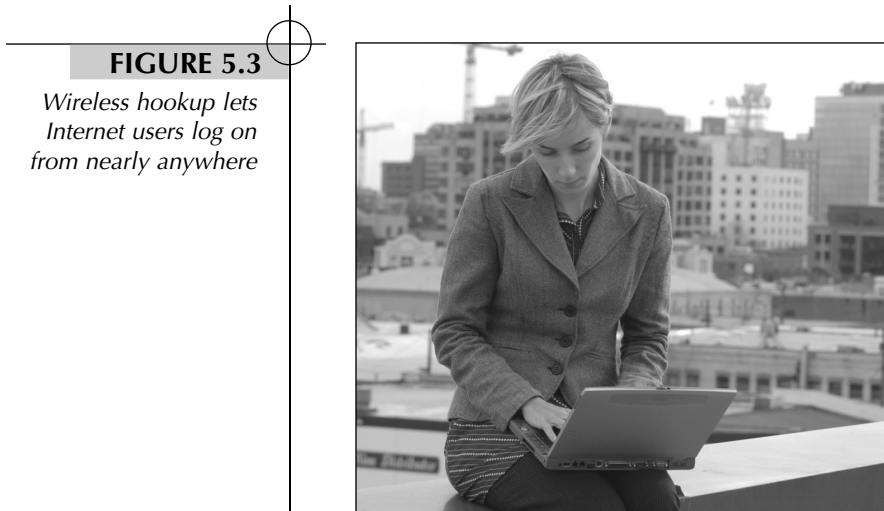
Complicated Figure Caption Usage

Designers have the ability to keep things simple, but also are well known for getting tricky. Their job is to make sure that the elements throughout have a harmonious ambiance to them. There is nothing worse to look at a design on a 4-color biology book and the chapter feels disconnected. The problem for the pager is that they have to sometimes tackle some complicated scenarios.

The next few examples are figure captions that require some advanced thinking and strategy to achieve. Most figure/caption combinations I have found can be through Xtags, but they do entail some thought ahead of time.

Side Element with 5 Grouped Boxes

This particular example is problematic initially because of the number of elements, but through some inventiveness we can make this less complicated. Here is the way it needs to appear:



This is comprised of five grouped boxes due to the fact that the vertical rule extending alongside the image needs to be a .5 pt rule. I would normally use the line feature for this, but cannot since it doesn't support relative placement. A 1 pt unanchored box will be used that requires .5 pt to be masked off.

To me, the masking of the vertical picture box is the only difficult element. The figure number can be achieved by using the above and below rules in Quark and setting both up in a style sheet. The Xtags style sheet codes for the rule above and below are:

```
<*ra(0.5,"Solid",K,100,0,-24,6.45)><*rb(13,"Solid",K,20,24,-10,-6.45)>
```

This makes that part easier without requiring two extra picture boxes. To start this, I will begin by trying to keep this as simple as possible. Here's the approach I would take with this figure:

1. The text box must start 15 pts from the top of the pasteboard to accommodate the vertical rule. It will also need to have a "None" background.

```
<&tbu2(0 B,15,10p,24p?,,,,,(,),(),n)>
```

2. The image just needs to be 1p to the right of the image with a 1.2 pt drop from the caption. The image needs to be trapped within the keyline.

```
<&pbu2(12 TR1,1.2,30p?-.5,48p?-.5,,,,,0.5,(n),(,100),,,,m,,, -0.5,-0.5,
,, "56 GB Disk:Xtags Book:Images:Chapter 04:fg04_011.eps" ,,,)>
```

3. Since we need a .5 pt. vertical line extending the length of the image, we will need to make a 1 pt box for this. We cannot use a vertical line for this because we need the line to be relative to the length of the image, which the lines feature does not currently support.

The problem we run into is that a picture box tag needs 1 pt minimum width. To make a .5 pt rule requires another 1 pt black 0% filled box to overlap it by .5 pt. This box also needs to extend above the image 1p2. Therefore, we need to make two relative box images.

The first needs to be read off of the image, and the second off of the first vertical picture box. The first box reads as follows with -2p1.5 TL1 from the image with -14 pts above the top of the image. We'll add 1p2 in the relative tag for the space needed in the vertical picture box to extend the full length of the image:

```
<&pbu2(-2p1.5 TL1,-14,1,(48p,R,1p2,0),,,k,,(n),(,100),,K,0,,m,,,,,"" ,,,)>
```

The second box reads off of the first box with -.5 pt TL1 from the image:

```
<&pbu2(-.5 TL1,0,1,(48p,R,0,0),,,k,,(n),(,100),,K,100,,m,,,,,"" ,,,)>
```

4. The final element is the circle that will read off of the black 1 pt vertical picture box with -8.2 TL1. This circle uses the "o" option added.

```
<&pbu2(-8.2 TL1,5.8,17,17,,,n,(0.5,o),(n),(,100),,n,,,m,,,,,"" ,,,)>
```

5. The 5 picture box grouping needs to be:

```
<&g(1,2,3,4,5)>
```

6. Together, this code will be:

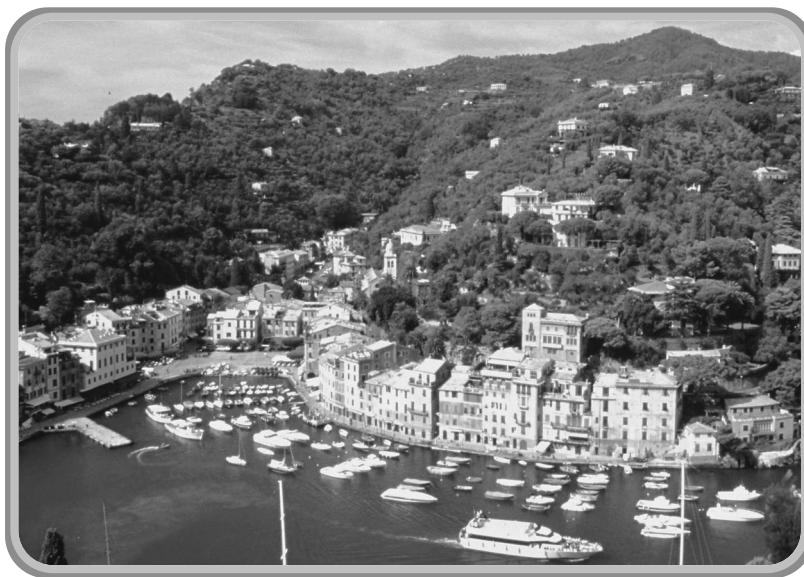
```
<&tbu2(0 B,15,10p,24p?,,,,,,(,),,(,),n)>@fgn:Figure 5.3
@fgc:Wireless hookup...anywhere<&te><&pbu2(12 TR1,1.2,30p?-
.5,48p?-.5,,,,,0.5,(n),(,100),,,,m,,, -0.5,-0.5,,, "56 GB Disk:Xtags
Book:Images:Chapter 04:fg04_011.eps" ,,,)><&pbu2(-2p1.5 TL1,-
14,1,(48p,R,1p2,0),,,k,,, (n),(,100),,K,0,,m,,,,,, " ,,,)><&pbu2(-.5
TL1,0,1,(48p,R,0,0),,,k,,, (n),(,100),,K,100,,m,,,,,, " ,,,)><&pbu2(-8.2
TL1,5.8,17,17,,,n,(0.5,o),(n),(,100),,n,,,m,,,,,, " ,,,)><&g(1,2,3,4,5)>
```

7. By using a translation table, this code would be broken down for the markup department in simplicity shown here:

```
[[f1]]@fgn:Figure 5.3
@fgc:Wireless hookup lets Internet users log on from nearly
anywhere[[f2]]Chapter 04:fg04_011.eps[[f3]]
```

Relative Double Border

This one is fairly easy, yet it requires two rules around it. The second box needs to be relative to the other picture box plus 4 pts. Here's an example of how the final imported image should appear.



Portofino, Italy is known for it's harbor that is full of boats.

This is one of those that can give someone trouble because they tend to overcomplicate it. The first obvious thing we see is that the picture box is rounded with a 25% 3 pt border around it. Around that is a 45% 3 pt border.

To get this effect, start by approaching this systematically:

1. Let's start by dealing with the image. Since the dark outer border needs to be placed second, we need for this to import 3 pts from the top left of the pasteboard. We also need a 3 pt rule with rounded corners, and a 1pt corner radius (which for this will be set at "22" and a 25% rule shade.

```
<&pbu2(3 B,3,42p?-.5,45p?-.5,,,n,(3,22,),(n),(25,),"Solid",K,0,,m,,
,-0.3,-0.3,,,"56 GB Disk:Xtags Book:Chapter 04:fg04_012.eps",,)>
```

2. The outer border needs to be -3 TL1 from the first box and an additional 6 points width and depth added to the relative placement:

```
<&pbu2(-3 TL1,-3,(42p,R,p6,1"),(45p,R,p6,1")>
```

3. The text box below is very standard, but is relative to the outer picture box's width. The full code for this box is:

```
<&pbu2(3 B,3,42p?-.5,45p?-.5,,,n,(3,22,),(n),(25,100),"Solid",K,0,
,m,,,-0.3,-0.3,,,"56 GB Disk:Images:Chapter 04:fg04_012.eps",,)>
<&pbu2(-3 TL1,-3,(42p,R,p6,1"),(45p,R,p6,1"),,,k,n,(3,28,),(n),
(45,100),"Solid",K,0,,m,,,,,,,"",,)><&tbu2(0 BL1,7,(42p,R,0,1"),
6p6?)>@f5:Portofino...of boats.<&te><&g(3,2,1)>
```

Caption Top, Source Line Bottom

At first glance, this appears very easy, but you'd be surprised on how difficult this can be to some pagers. I've actually seen them bring the source into the figure caption at the top and then create a text box below the image and link to it. Upon the linking, they readjust the spacing, which is extra work.

FIGURE 1.1 San Francisco's Golden Gate Bridge is one of America's most popular landmarks.



Source goes here

The way to approach this without the hand manipulation is:

1. Start with a text box bringing in the caption followed by a picture box. Just like the standard top caption positioning.
2. Following the picture box, have another text box start. This will be coded as follows:

```
<&tbu2(0 B,0,24p,6p?)>@fgc:<@fgn>Figure 1.1<\f><\f><\i><@$p>San
Francisco's ....<&te><&pbu2(0 BL1,10,24p,50p?,,,n,,(n),,K,0,,c,,
,,,,,"56 GB Disk:Xtags Book:Images:Chapter 04:fg04_013.eps",,,)>
<&tbu2(0 BL1,7,(24p,R,0,0),4p?)>@fgs:Source...<&te><&g(3,2,1)>
```

3. When using a translation table, the markup department will code the text like this:

```
@fgc:[[f1]]Caption Text[[f2]]Chapter 04:fg04_013.eps[[f3]]
@fgs:[[f4]]Source Text[[f5]]
```

4. I know you are thinking “this is so easy, why even show this?” There’s an extra step that is important. On many images there will not be a source line. You do not want Xtags bringing in an empty text box. The best thing to do in this situation is to have the markup department still write up the source line, but leave the text blank afterwards if a source doesn’t exist. Then run an AppleScript that would do the following.

```
tell application "BBEdit"
    activate
    replace "\\r\\@fgs:\\[\\[f4]]\\[\\[f5]]" using "<\\&g(1,2)>"
        searching in text 1 of text document 1 options {search
            mode:grep, starting at top:true, wrap around:false,
            backwards:false, case sensitive:true, match words:false,
            extend selection:false}
    end tell
```

7. By doing this, the coding department will not have to try and change the Xtags for those images. After running the AppleScript, the coded file will look like this:

```
@fgc:[[f1]]Caption Text[[f2]]Chapter 04:fg04_013.eps[[f3]]<&g(1,2)>
```

Working with Blends in Captions

I can hear you saying to yourselves, “Xtags doesn’t support blends.” You are correct about that. Well at least at press time this was a true statement, but there is a way around this by using relative placement and having a blend already created in Photoshop or Illustrator.

Here's an example of an image that would require a blend in the caption:



Figure 2 This is the caption with the upper right blend behind it that will need to shrink to fit.

By looking at the above treatment, we see that the caption is top right and the blend is behind the caption with a 6 pt inset. So how do we achieve this?

1. Build the blend in Photoshop or Illustrator. Make sure it is deep enough to accommodate the longest caption. For this example, I thought it might be 18p maximum depth.
2. The *First Baseline* on the caption will be "1p2". There will also need to be an extra 6 points depth on the text box, so add p3 to the depth field.

```
<&tbu2(12 TR1,0,12p,15p?+p3,,,n,,(n),(,100),,n,0,,,,,1p2,,,,)>
```

3. The blend will use a picture box at -12p TR1. This is the length of the caption. This will also need a relative placement on the depth. The length is obviously going to remain the same.

```
<&pbu2(-12p TR1,0,12p,(20p,R,0,.25")
```

4. The full code will be as follows:

```
<&pbu2(0 B, 0,25p?,24p?,,,n,,(n),(,100),"Solid",K,0,,m,,,,,"56 GB  
Disk:Xtags:fg04_015.eps",,)><&tbu2(12 TR1,0,12p,15p?+p3,,,n,,(n),  
(,100),,n,0,,,,,14,,,,)>@FIG:<@FIGN>Figure 2<@$p><\f>Caption.<&te>  
<&pbu2(-12p TR1,0,12p,(20p,R,0,1"),,,k,n,,(K,n),(100,100),"Solid",  
K,,,m,,,,,"56 GB Disk:Xtags:blend.tif",,"")><&g(1,2,3)>
```

5. If the caption is needed on the opposite side of the image, remember to rotate the image "blend.tif" 180° in the fifth field. You also need -13p TL1 for the caption start to account for the 1p space between the image and the caption. The full code will be:

```
<&pbu2(14p B,0,34p?,26p?,,,n,,(n),(,100),"Solid",K,0,,m,,,,,"56 GB  
Disk:Xtags:fg04_015.eps",,)><&tbu2(-13p TL1,0,12p,15p?+p3,,,n,,  
(,n),(,100),,n,0,,,,,14,,,,)>@FIG:<@FIGN>Figure 2<@$p><\f>Caption.  
<&te><&pbu2(0p TL1,0,12p,(20p,R,0,.25"),180,,k,n,,(K,n),  
(,100),"Solid",K,,,m,,,,,"56 GB Disk:Xtags:blend.tif",,"")><&g(1,2,3)>
```

6. The image and caption would appear as seen below:

Figure 2 This is the caption with the upper right blend behind it that will need to shrink to fit with the blend. Even if it goes to more lines.



7. When using a translation table, the markup department will code the text like this:

```
@FIG: [[f1]]Chapter 04:fg04_015.eps[[f2]]Figure caption[[f3]]
```

If needing both a verso and recto version, you could have a script duplicate each figure. By having the verso and recto translation table definitions, the verso figure could be coded as:

```
@FIG: [[f4]]Chapter 04:fg04_015.eps[[f5]]Figure caption[[f6]]
```

Adding a Number on the Figure

This is seen more often in home repair books, but I've found that this is typically something that happens in QuarkXPress rather than on the actual image. This is where there are illustrated step-by-step instructions showing what order to do the work in. Since the author may, at some point, add a step, they do not want the number to be part of the image. That is why it's the pager's responsibility to group this with the image. Here's an example:



As you can see, the number lands in the bottom left corner with a white rule around it. However, the white rule is only on the top and the right of the 20% gray numbered box. Here is how we would accomplish this:

1. Bring in the image, but make sure there's a slight adjustment on the image. I put a $-.25$ on both the width and depth of the image to make sure there is no extra white space around the image.

```
<&pbu2(0 B,0,35p?-.25,40p?-.25
```

2. The box with the number in it will need to be positioned at the bottom left corner with a $-1p6.5$ y coordinate. The depth of the box is $1p6.5$, so this will fit in perfectly.

```
<&tbu2(0 BL1,-1p6.5,1p2.5,1p6.5
```

3. We are now going to incorporate two lines into this. This is where you'll get to use some ingenuity to make this happen. The first line needs to position at the top left "0 TL1 x" and "0 y" of the numbered text box.

```
<&lbu(0 TL1,0,0,14.5,or,,1,W,,,,)>
```

4. The second line needs to read off of the bottom right of the numbered text box. That is why we are calling out "BR2" with 0 for the x and y.

```
<&lbu(0 BR2,0,90,19,or,,1,W,,,,)>
```

5. The full Xtags code for this will be:

```
<&pbu2(0 B,0,35p?-.25,40p?-.25,,,,n,,(n),(,100),,,,,m,,,,,"56 GB  
Disk:Xtags:Chapter 04:fg04_014.eps",,,)><&tbu2(0 BL1,-1p6.5,1p2.5,  
1p6.5,,,,n,,(n),(,100),,K,20,,,,14,,,,)>@label:1<&te><&lbu(0 TL1,0,0,  
14.5,or,,1,W,,,,)><&lbu(0 BR2,0,90,19,or,,1,W,,,,)><&g(4,3,2,1)>
```

6. This could also be handled with a rule around the numbered box, but the white border would be 1 pt below the image and to the left of the image that would not be visible. The box width would need to be an extra 1 pt in width. The code for the text box would instead be:

```
<&tbu2(-1 BL1,-1p8.5,1p3.5, 1p7.5,,,,n,1,(W,n),(,100),,K,20,,,,14,,,,)>
```

Rotated Solid Shadow on Image

This is to illustrate how some images are handled where the designer will put a decorative background behind the image to give the feel of a shadow. I've seen it before where the image is actually rotated 4 or 5° to give it a different effect. This is very easy to do by using relative placement for the width and depth of the actual image.

Here is how the image and caption will import:

FIGURE 1.1

Many cities bordering the oceans have waterways cut through the city.



The method to get this effect is as follows:

1. The text box with the caption is standard with the image falling to the top right of the caption.
2. The shadowed box will need to be the exact width and depth of the image with a 4° rotation. This also needs to be sent to back.

```
<&pbu2(0 TL1,0,(25p,R,0,0),(40p,R,0,0),4,,k,
```

3. The full code will be:

```
<&tbu2(10 B,10,82,61.379,,,,n,,(n),(,100),,,,,,,) >@fgn:Caption<&te>
<&pbu2(12 TR1,0,25p?,40p?,,,,,n,,(n),(,100),,,,,m,,,,,, "56 GB Disk:
Xtags:Chapter 04:fg04_016.eps",,,)><&pbu2(0 TL1,0,(25p,R,0,0),
(40p,R,0,0),4,,k,n,,(n),(,100),,K,85,,m,,,,,, "",,,)><&g(3,2,1)>
```

Shadow on Images Shrinking-to-Fit Both Dimensions

Here is a very advanced approach, but it is something I figured out and I really think it can be a time saver. There are several shadow creating XTensions on the market that will create an image to the size of your image, then saves the shadow out as a Photoshop file. This can sometimes be a timely process, especially if you are working on a marketing book with 40 to 50 images.

What if you could have the shadow that would appear on every image without having to do the manual work after the import? I have discovered a way to do this. The main question is if you can work with shadows with a different aspect ratio. For example, would it be acceptable for the x to be 55% and the y to be 67%. This is very hard to detect with the eye and I feel that they look very acceptable.

Here's an example of an image with the type of shadow I'm referring to:



I have two shadows that I use for this. The first is set horizontally and is known as "Shadow2.tif". The second is set vertically and is "Shadow1.tif". This way the scaling will not be so varied that major banding occurs. To get this effect:

1. The image pulls in standard with only a -p.2 width and depth image reduction to assure that no white space will occur.
2. Using relative placement, I shrink to fit the shadow to the same proportions as the original image using relative placement for both the width and depth, but offset by 7 pts top and left. In the #16 placement field, we will be using the "f" which is the "expand" or "shrink-to-fit" option. This will fit the shadow, but will scale it to fit where it could be 57% x and 72% y.

```
<&pbu2(7 TL1,7,(36p2,R,p.5,1"),(51p4,R,p.5,1"),,,k,n,,(K,W),
(100,100), ,K,0,,f,,,,,"56 GB Disk:Shadows:Shadow2.tif",,"")>
```

3. When coding, it is essential to know the dimensions of the images. You will need to know which images are horizontal and which are vertical. Scripts can be written to accomplish this if you don't use an art log. Once you can determine this, the coding will need to reflect this by marking up the text file like this:

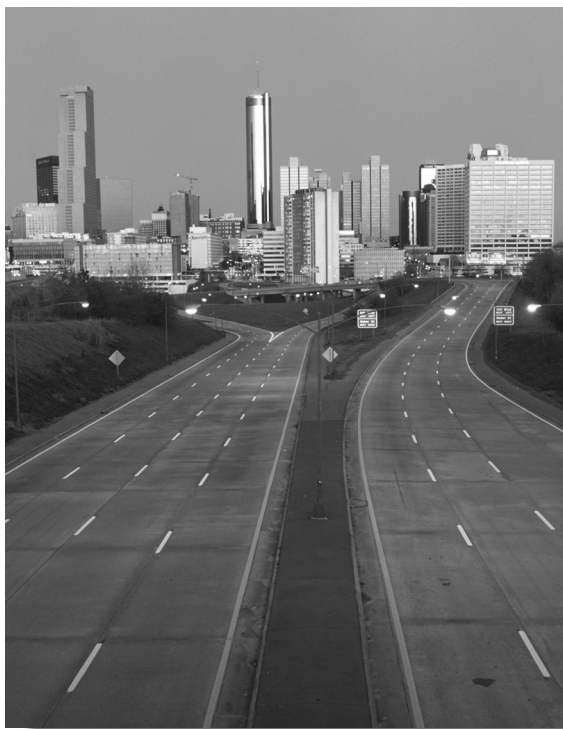
```
[[f1]]Chapter 01:fg01_001.eps[[f2h]]      for horizontal
[[f1]]Chapter 01:fg01_002.eps[[f2v]]      for vertical
```

92 *Xtags Maximized*

4. The full codes will be written in the translation table as:

```
[[f1]]      <&pbu2(10p B,0,37p?-p.2,51p4?-p.2,,,n,,(K,W),(100,100),
,K,0,,m,,,,,"56 GB Disk:Images:"
[[f2h]]     " ,")><&pbu2(7 TL1,7,(36p2,R,p1,1"),(51p4,R,p1,1"),
,,k,n,,(K,W),(100,100),,K,0,,f,,,,,"56 GB Disk:Xtags
Book:Shadow Folder:Shadow2.tif",")><&g(1,2)>
[[f2v]]     " ,")><&pbu2(7 TL1,7,(36p2,R,p1,1"),(51p4,R,p1,1"),
,,k,n,,(K,W),(100,100),,K,0,,f,,,,,"56 GB Disk:Xtags
Book:Shadow Folder:Shadow1.tif",")><&g(1,2)>
```

The horizontal image will then pull in as:



5. The proportions on the shadow are 47.9% and 39.2%.

Unless you have a problem with the horizontal scaling causing some problem during the printing of the file, this should be acceptable. Before adapting this type of a workflow, it's a good idea to run some tests and see if anyone has a problem with the quality of the shadows. I find that the quality doesn't seem to suffer any as long as you use the horizontal shadow or the vertical shadow that are more proportionate to their images.

Screened Caption Under Image

This type of treatment is seen more often in gardening and cooking books, yet it could be used in textbooks as well. It's when you have the caption screened underneath the image. Look at this example:



FIGURE 15.3 Wireless Internet

With the onset of online shopping, auction companies, and the ease of use through wireless Internet, home business ventures are growing to be where some of the big money lies. Once established, setting priorities and hours are easier, but still require a lot of push.

The idea here is that the caption screen will always be 6 points down from the top of the image and 6 points from the bottom of the image. So regardless of the length of the caption (unless it goes longer), the screen will always be that depth. To do this:

1. The picture box will need to start 6 points from the left, but would need a (2,8,6,8) runaround. If the image needed to be on the right, you could just move the image 6 points from the right and the caption would still break the same without adjusting the styles, as shown here.

FIGURE 15.3 Wireless Internet

With the onset of online shopping, auction companies, and the ease of use through wireless Internet, home business ventures are growing to be where some of the big money lies. Once established, setting priorities and hours are easier, but still require a lot of push.



2. The text box will come in next based on the depth of the image as well as the runaround from the image. There is a 16 point relative placement depth reduction so the caption will be 8 points offset from the top and bottom.

<&tbu2(-6 TL1,8,29p,(40p,R,-16,0),,,k,n,,(n),(,100),,K,15,,,,,17,,,,,>

3. The complete code for this will be:

```
<&pbu2(6 B,0,30p?-25,40p?-25,,,,,1,(n),(15,100),,n,,(2,8,6,8),m,,
,-.5, -.5,,,"56 GB Disk:Xtags Book:Images:Chapter 05:ch05_011.eps",,,)>
<&tbu2(-6 TL1,8,29p,(40p,R,-16,0),,,k,n,,(n),(,100),,K,15,,,
,17,,,,,)>Figure Caption here<&te><&g(2,1)>
```

Screened Caption Below Image

The screened handling of the caption could also be used where the image was above the caption as shown here:



FIGURE 15.3 Wireless Internet

Online shopping and auction companies, have risen in popularity due to the ease of use through wireless Internet.

The big difference compared to the previous page is that the image would not need a runaround because the text box could start below the image. The bar above the image could be its own separate picture box, but would still need to be using relative placement because it would need to have a depth reduction of 1p.

```
<&pbu2(6 TL2,-6,(30,R,-12,0),6.1,,,k,n,,(n),(,100),,K,15,,m,,,,,"" ,,,)>
```

Here is the code for the image and caption:

```
<&pbu2(0,6,30p?-p.25,32p?-p.25,,,,,n,1,(n),(15,100),,n,,,m,,,,,"56 GB
Disk:Xtags Book:Images:Chapter 04:fg04_019.eps",,,)><&tbu2(6 BL1,
-1,(30p,R,-12,0),22p?p3,,,k,n,,(n),(,100),,K,15,,,17,,,,,)>Figure Caption
Here<&te><&pbu2(6 TL2,-6,(30,R,-12,0),6.1,,,k,n,,(n),
(,100),,K,15,,m,,,,,"" ,,,)><&g(3,2,1)>
```

Fit-to-Height and Fit-to-Width Multi-Piece Example

In chapter 3, we discussed the *Fit-to-Height* and *Fit-to-Width* option that can be included in the width and depth fields. This is one of those options that I can not rave about enough. There are so many applications for it and can really make fitting images to the area you need an easier task than ever before.

I have had situations in books where the author wanted to include 3 images together set up similar to this example:

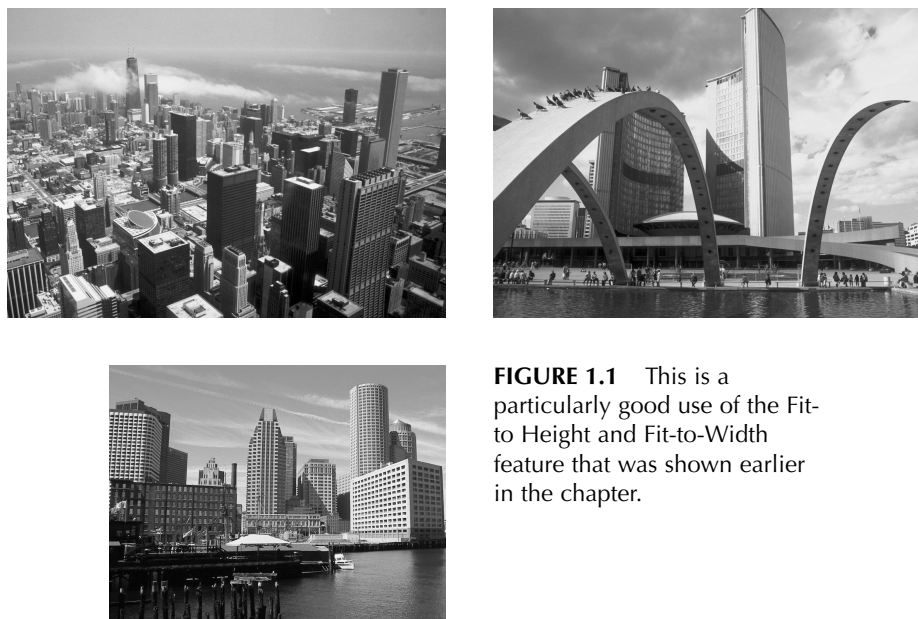


FIGURE 1.1 This is a particularly good use of the *Fit-to-Height* and *Fit-to-Width* feature that was shown earlier in the chapter.

The problem with a layout like this is you want it to have some balance, but you aren't sure the depth or width of the images based on the text area. If the book is a modified pickup, the margin area may have changed from the previous edition. This can be solved handling the images with the *Fit-to-Height* and *Fit-to-Width* option. The images need to have a balanced feel, so using the top right and bottom right positioning is essential for this to work. Here is how I approached the example above:

1. In this case, the two top images each need to have a 9p8 depth. To achieve this, in the #4 field, you will need to put (9p8,f) in both picture box tags. The picture box to the right will need to have an 18 point top right placement as well. Here's an example of the code for the top right image:

```
<&pbu2(18 TR1,0,35p?,(9p8,f),,,,n,,(n),(,100),,,,m,,,,,
,"56 GB Disk:Xtags Book:Images:Chapter 03:fg03_018.eps",,,)>
```

2. The bottom image needs to have a maximum width of 10p6. The depth is not an issue, so in the #3 field, we will need to have (10p6,f). This box will also position off of the picture box directly above it. This will need to be -10p6 bottom right (-10p6 BR2) with an 18 point y offset for this to position correctly.

```
<&pbu2(-10p6 BR2,18,(10p6,f),35p?,,,n,,(n),(,100),,,,m,,,,,
,"56 GB Disk:Xtags Book:Images:Chapter 03:fg03_019.eps",,,)>
```

3. The text box will read off of the top right picture box where it requires a "0 BL2" with an 18 point y offset. It will also need to be 10p6 to match the width of the bottom left image for balance.
4. The complete code is as follows:

```
<&pbu2(0 B,0,30p?,(9p8,f),,,,n,,(n),(,100),,,,m,,,,,"56 GB Disk:Xtags
Book:Images:Chapter 03:fg03_011.eps",,,)><&pbu2(18 TR1,0,
35p?,(9p8,f),,,,n,,(n),(,100),,,,m,,,,,"56 GB Disk:Xtags
Book:Images:Chapter 03:fg03_018.eps",,,)><&pbu2(-10p6
BR2,18,(10p6,f),35p?,,,n,,(n),(,100),,,,m,,,,,"56 GB Disk:Xtags
Book:Images:Chapter 03:fg03_019.eps",,,)><&tbu2(0
BL2,18,10p6,12p?,,,n,,(n),(,100),,n,,,,,,)>All box text
here<&te><&g(1,2,3,4)>
```

Many possibilities can happen with the *Fit-to-Height* and *Fit-to-Width* option. This is a good feature to practice with. Try different scenarios to see what you can come up with.

In Closing

While this is a very good sampling of what can be done with the figures and captions, it is just scratching the surface. The next chapter is going to look at boxes. We will be using techniques we used in this chapter, while throwing in some even more advanced features to go beyond what many might consider the boundaries of Xtags.

5

Creating Boxes and Elements

CHAPTER

The past couple chapters have given you some background on what can be done with the text box, picture box, and line tags. Xtags offers so many options that practically any type of box, from basic to complicated, can be created if you take the time and develop an approach. I have rarely seen a floating element that I could not accomplish using Xtags. The most difficult situations have been boxes that exceed one page. Most others can be accomplished with a strategy on where to begin. The goal when building boxes is being able to use them many times throughout the project, so there needs to be consistency.

The most obvious thing to consider when you are building a box is to ignore the fact if the element is in color to begin with. I've witnessed pagers who see a lot of color and just freeze. I tell them to ignore the color at first and just deal with what it is going to take to reproduce this. That, to me, is where a person really needs to begin.

Try to keep the boxes as simplified as possible to avoid a lot of hand manipulation afterwards. Just because a designer has an element set up a certain way does not mean you have to keep it the way they originally created it. In most cases, a designer is not a pager. Therefore, they do not look for the best way to handle it. They are looking at the box for its appearance, whereas a pager will see it for its functionality.

As I mentioned earlier, you should always begin by creating your box and doing a *Copy Xtags Text* and *Paste* combination to see which codes are making up your box. In most cases, the order will not be what you want, there isn't any relative placement, and other factors will not be exactly what you want. What it does do is gives you a starting point to begin with.

Origin Tags

When doing the *Copy Xtags Text* and *Paste* on a group of boxes, you will notice an *Origin Tag* preceding the text and picture boxes. It looks like this:

```
<&o(44,384)>
```

This is showing the x and y location of the grouped boxes on the actual page. I typically delete these because I use Autopage for the actual positioning of the floating objects. For some workflows, they may serve more of a purpose because it is much easier to set the 0x, 0y for your first picture or text box rather than the actual page positioning. That's where the *Origin Tags* come in.

Let's consider a chapter opener with an opening photo on the page. You may find that if you use Autopage, this first image will not be placed because your chapter opener is typically set up different than the rest of the document. So, in order for this image to place and not remain on the pasteboard, the *Origin Tags* tags could be used for proper placement of that image. Look at this chapter opener example of how the image will be placed:



Traveling and commuting is a great time to get freelance work or other business done.

The chapter opener page needs to have the opening image placed in the center of the page in that exact x and y coordinate. You could bring in that image by hand, but it would be easier to do the followings.

1. Set up the opening photo to pull in through the translation table. We know that the exact position of this image is 4p9(x) and 14p9(y). This means that the *Origin Tag* needs to be:

<&o(4p9,14p7)>

2. The picture box and text box tags can be set as normal with one exception. You cannot use the pasteboard specifier (0 B, 0) because it needs to be pulled on the page. Instead just have the tag read like this:

<&pbu2(0,0,40p?,52p?)>

3. The full tag will be:

```
<&o(4p9,14p7)><&pbu2(0,0,40p?,52p?,,,n,l,(n),(,100),,,,m,,,,,"56
GB Disk:Xtags:Images:Chapter 05:ch05_001.eps",,,)><&tbu2(0 BL1,
9,40p,7p?,,,n,,(n),(,100),n,,,,,,,>@fgc:Caption<&te><&g(2,1)>
```

3. As long as the art tags are called out on this page, the image will place in that exact location with each chapter.

This example proves that the *Origin Tags* can be used to save from doing manual tasks. If you can think of something that you know needs to be on a certain page in a certain position, this will make that task easier to accomplish.

Pre-paging with Origin Tags

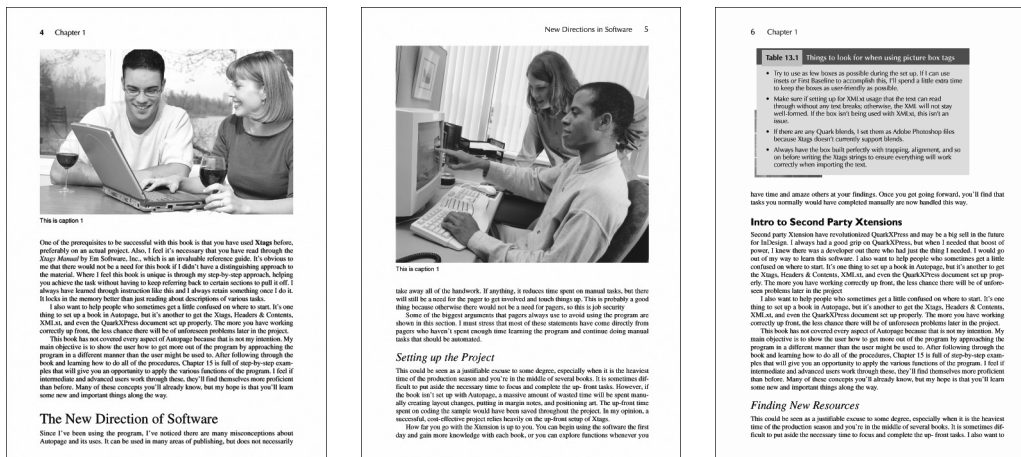
The *Origin Tags* can serve another purpose that I find very useful when first importing the text and images. For Xtags users who do not currently use Autopage, the *Origin Tags* can serve as a way to place the images throughout the document where they are referenced. This can be accomplished by just specifying the first image's *x* and *y* *Origin Tag* and having the coder type the Xtags string at the reference point. The remaining *Origin Tags* would need to be set at 0 *x*, 0 *y*. Here's an example of how the translation table would need to be set. Notice the runarounds on the `[[f1]]`, `[[f2]]`, and `[[t1]]` tags.

```
[[o1]] <&o(5p3,5p6)>
[[o2]] <&o(0,0)>
[[f1]] <&pbu2(0,0,(31p6,F),35p?,,,i,,(n),(,100),,K,0,(12,12,6,12),m,,
,,,,,"56 GB Disk:Xtags Book:Images:
[[f2]] ",,)><&tbu2(0 BL1,6,31p6,5p?,,,i,,(n),(,100),,n,,(1,1,24,6),,,
,,,,,)>
[[t1]] <&tbu2(5.5 TL1,6.5, 24p6,30p?p.5,,,i,,(n),(,100),,n,,
(24,34,34,94),,,,,,)>
```

This is specifying that when using the `[[o1]]` *Origin Tag*, that the image will place at 14p6 *x* and 5p6 *y*. This is the text area where the image will need to place. I will only use `[[o1]]` tag once. The remaining images will have the `[[o2]]` specifier which is saying that all remaining images need to be set at the same *x* and *y* coordinates as the first image, but on the page they import on. By using a runaround, the images, tables, or boxes will place throughout the document on the page where they were originally typed. The only downfall to this that I have found, is if more than one image falls on the same page, they will stack on top of each other. If you are not careful, this is an easy way to lose an image.

I do want to point out that an argument could be that instead of using the *Origin Tags*, that you could just include the page area *x* and *y* coordinates in the Xtags strings. This makes certain the images would always pull into that location rather than using the *Origin Tags*. This is acceptable to do as long as you do not want to use the same strings in different places such as art that would fall within boxes or tables. The *Origin Tags* give you much more control over when you want to use the coordinates for placing the images.

Here is an example of how these 3 pages will layout during the import process when using the *Origin Tags*. I used the *Origin Tags* on the images and table.



The layout is achieved by having a runaround on the art, tables, and captions so the text is pushed forward. In the coded file, the *Origin Tags* will need to be called out before the Xtags strings. For this example, I used a translation table and the `[[o1]]` and `[[o2]]` specifiers. The tags will be set in the text file as:

```
[[o1]][[f1]]Chapter 05:fg0101.eps[[f2]]@fgc:This is caption 1[[f3]]
[[o2]][[f1]]Chapter 05:fg0102.eps[[f2]]@fgc:This is caption 2[[f3]]
[[o2]][[t1]]Table text goes here[[t2]]
```

As I pointed out, this doesn't make a flawless paged document, but it saves a lot of initial time in placing the images near their referenced location. The Xtags string does have to be located near the referenced text. For example, if you have Figure 1.1, The text file should be set up like this:

```
@textstyle:This can be seen in Figure 1.1.
05:fg0101.eps[[f2]]fgc:Figure 1.1 Caption
```

The same would go for a table or box. If the table is called out, it should be similar to:

```
@textstyle:See the data in Table 1.1
[[o2]][[t1]]Table text 1.1 here
```

Most coders tend to type all of the figures, tables, and boxes at the end of the text document. I suggest following this in most cases, but if trying to handle your floating elements with the *Origin Tags*, this would not be the ideal situation. After the initial import, it's important to look through *Picture Usage* to be sure images are not stacked on top of each other.

Although there are some issues with doing this, it does make for a more expedited paging experience on a book with many images. One definite issue

is that if the inside and outside margins are not equal, the image on the verso and recto pages will position differently. The idea here is to get the images close to the callout with only minor adjusting needed. If you are hoping to get the images to position exact, you will want to invest in Autopage, since this is one of the many features of the program.

Creating Boxes with Xtags

Creating boxes with Xtags is one of my favorite aspects of the program. This is where one can really push the envelope and accomplish some intricate things. In this section, I have created many examples to show you the time-saving benefits of building boxes through Xtags code from simple boxes to some very involved scenarios.

From experimentation, I can take virtually any box and write the code to have it import using Xtags. The advantage to this is that everything will trap (if set up correctly) and shrink-to-fit, leaving very little, if any, adjustments to be made when paging.

I have a few procedures I consistently follow when setting up boxes:

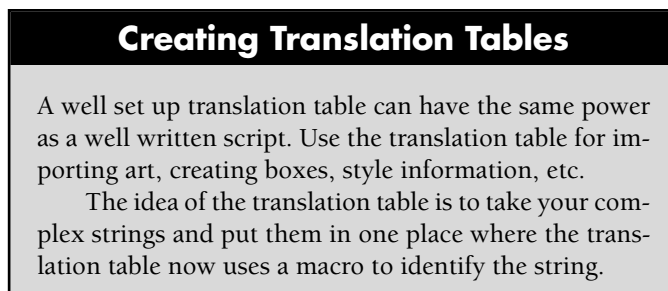
1. Try to use as few boxes as possible during the set up. If I can use insets or *First Baseline* to accomplish this, I'll spend a little extra up-front time to keep the boxes as user-friendly as possible.
2. Make sure if setting up for XML usage that the text can read through without any text breaks; otherwise, the XML will not stay well-formed. If the box isn't being used with XML, this isn't an issue. I still try to make it as simplified as I can.
3. If there are any blends needed, I set them as Adobe Photoshop or Adobe Illustrator files because Xtags currently does not support Quark blends.
4. Always have the box built perfectly with trapping, alignment, and so on before writing the Xtags strings to ensure everything will work correctly when importing the text.
5. Make sure to set up as much as possible in the style sheets. This includes any rules.

While there are so many things that can be accomplished with boxes, I feel that this section will open up your mind to what Xtags can do. For Autopage users, it is acceptable to use the library and the T=E code, but you'll find that a lot less hand manipulation happens when importing with Xtags. I'll let you make your decision after reading through this chapter.

Single-Column Box

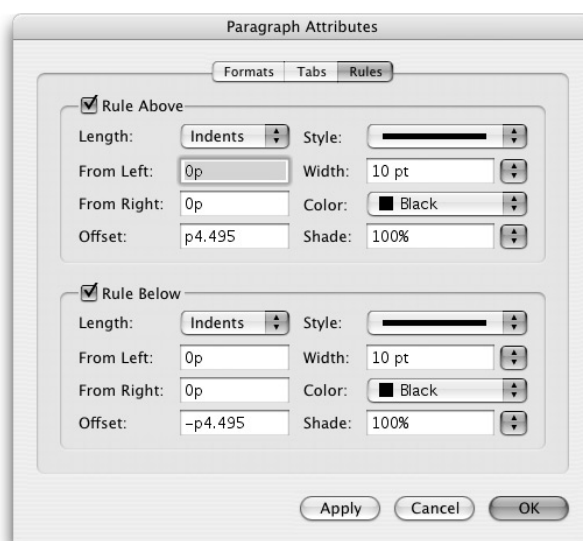
This is a fairly uncomplicated box that many less-experienced pagers may set with two or three boxes. In most cases, they would begin by first taking the

“Creating Translation Tables” title and create that box, then make the gray screen a picture box, and the text would be in its own box.



This can be completed with only one box and imported easily with Xtags. A few things can be accomplished in the Quark file:

1. The *Creating Translation Tables* head will need to be set up with Quark rules with “Rule Above” and “Rule Below”:



2. There will only be a need for a bottom inset in this example, so the styles will need left and right formats built in to the style sheets.
3. The multiple inset will need to be set with all zeros except the bottom, where you'll need 6 pts. This can be set up in the Text Inset field (#18) as (0,0,6,0).

The Xtags to create this box will be:

```
<&tbu2(0 B,0,21p,40p?,,,,n,1,(,n),(,100),,K,15,,,,(0,0,6,0),,,,)>Box Text here<&tc>
```

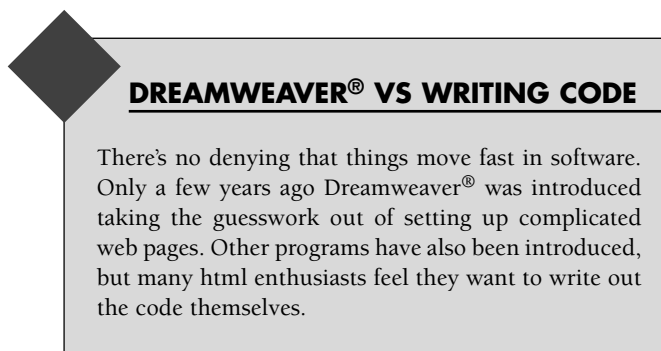
In the input file using a translation table would be:

```
[[t1]]@BOX_TTL:Creating Translation Tables
@T1:A well set up ...
@T:The idea ... string. [[t2]]
```

Single-Column Box with Rotation

This box may appear very basic, but it contains four items that make it unique, which are highlighted in the Xtags strings.

1. The black diamond box is rotated, so “45” is used in the box angle field to represent the 45° angle.
2. The text box was *Sent to Back* using “k” in the *flags* field.
3. The baseline offset is set for “2p” to keep from having to use an inset since the styles have 1p left and right indents.
4. As an alternative to using an inset for the bottom of the box, I added 5 points to the depth of the “shrink-to-fit” (16p?) on the text box, which added enough space to have 1p around the box.



This shows different options using Xtags. For this box, it's not necessary to use a BL1 since the boxes will always start in the same locations.

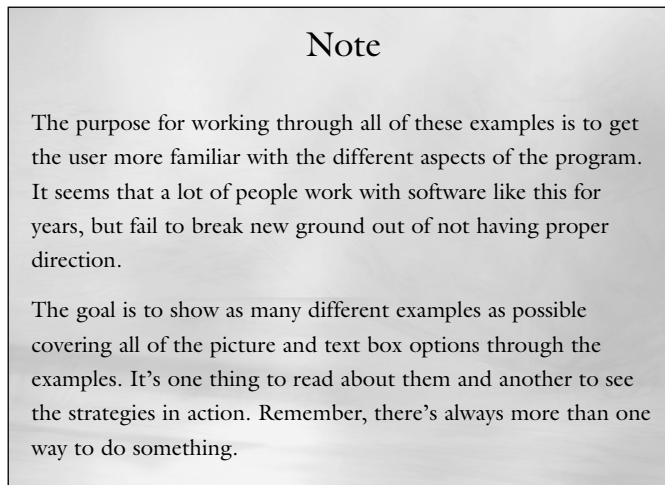
```
<&pbu2(6.3 B,6.3,30,30,45,,n,,(,n),(,100),"Solid",K,75,,m,,,,,"",
,)><&tbu2(21 B, 10.5,17p10,16p?p5,,k,n,0.5,(,n),(,100),"Solid",K,15,
,,,2p,,, )>@SUPTTL: Dreamweaver<+>®<$> vs Writing Code
@BOX:There's no denying ... themselves.<&te><&g(2,1)>
```

To reduce time again, it is a good idea to put this into a translation table so the coding will be easier. The markup department would only have to type the following:

```
[[t1]]@SUPTTL:Title
@BOX:Text[[t2]]
```

Simple Box with Background Art

This is a good example because it demonstrates how to put an art background behind a standard block of text. I've seen a similar box throw off some paggers when they try to set it up. The odd thing is that it isn't really that difficult. It all comes down to proper use of relative placement. Here's an example of how this should look:



To get these results, follow these steps:

1. The text box has a multiple text inset, but the rest is standard.
2. The picture box containing the image is where most of the work will be. This box needs to have relative placement (exact to the depth and width), has a .5 point rule around it, uses the "Send to Back" *flag*, and needs to position 0 TL1. The top left positioning is how it expands perfectly to fit the content. Here is an example of the picture box code:

```
<&pbu2(0 TL1,0,(21p,R,0,0),(24p,R,0,0),,,k,n,.5
```

3. The final code to get this result is:

```
<&tbu2(0 B,0,21p,24p?,,,n,(,n),(,100),,n,,,,(9,9,11,9),,,,,)>@ttl:Note
@nt:Text<&te><&pbu2(0 TL1,0,(21p,R,0,0),(24p,R,0,0),,,k,n,.5,(,n),
(,100),,,,m,,,,,"Disk 1:CH 05:fg004.eps",,,)><&g(1,2)>
```

4. The translation table would contain these two parts:

```
[[t1]] <&tbu2(0 B,0,21p,15p?,,,n,(,n),(,100),,n,,,,(9,9,11,9),,,,,)>
```

```
[[t2]] <&te><&pbu2(0 TL1,0,(21p,R,0,0),(24p,R,0,0),,,k,n,.5,(,n),
(,100),,,,m,,,,,"Disk 1:CH 05:fg004.eps",,,)><&g(1,2)>
```


Example of a Two Box Approach with Relative

I thought this would be a good example to demonstrate rounded corners and how to offset slightly using relative placement.

The approach to take on this box is to keep it to two boxes if possible. When first looking at this, it appears that this will not be possible, but I assure you it is. The main thing to look for is what can be put into the style sheets. That is always the best approach to reduce the number of boxes. With this initially I see that the title's bar can be built into the style sheet.

Here's how the box will look when imported into QuarkXPress.

NOTE

The purpose for working through all of these examples is to get the user more familiar with the different aspects of the program. It seems that a lot of people work with software like this for years, but fail to break new ground out of not having proper direction. The goal is to show as many different examples as possible covering all of the picture and text box options through the examples.

To get this to work requires following these steps:

1. Make sure you have the title's background shade built into the rules. Here is the style sheet code that makes up the rules:

```
<*ra(11,"Solid",K,100,28,28,3.6)><*rb(11,"Solid",K,100,28,28,-3.6)>
```

2. There needs to be a multiple text inset of (0,12,10,12) on the text box.

```
<&tbu2(0 B,0,21p,30p?,,,n,,(n),(,100),,n,,,,(0,12,10,12),,,,,)>@ttl:Note
```

3. The picture box around the text needs to position 0 TL1, but with an 11 point y coordinate. It also will need to be sent to the back, contain a .5 point rule, and will need relative placement with a 11 point depth reduction. There is also a slight corner radius. Here is the picture box code:

```
<&pbu2(0 TL1,p11,21p,(30p,R,-p11,0),,k,n,(0.5,12,),
(n),(,100), ,n,,,m,,,,, "" ,,,)>
```

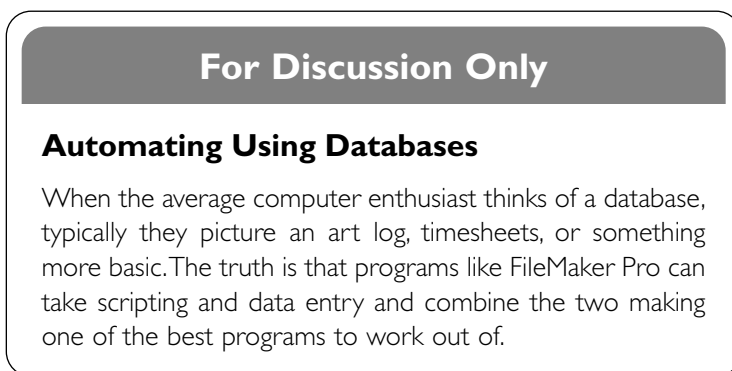
4. The complete code will be:

```
<&tbu2(0 B,0,21p,30p?p1,,,n,,(n),(,100),,n,,,,(12,10,12),,,,,)>@ttl:Note
@nt:Text:<&te><&pbu2(0 TL1,p11,21p,(30p,R,-p11,0),,k,n,.5,(n),(,100),
,n,,,m,,,,, "" ,,,)><&g(1,2)>
```

Rounded Corner Box with Unsupported Shape

This example displays a basic box, but there is something in this that Xtags does not currently support. The title has a box behind it that has rounded corners at the upper left and right, but the bottom left and right are squared off.

Most intermediate level users may feel that they cannot accomplish this because Xtags does not support this shape. However, this is where you need to use some ingenuity. Where you need to start is by first realizing that the power here lies in the “Send to Back” feature. The question is “how do we do this?” The answer, surprisingly, is much easier than you would think. Here is how the finished box should look:



1. Your text area will need to be by 23p. You'll need a p9 corner radius on the text box. There will also need to be a 2p1 First Baseline. An additional 8 points will be needed on the depth. Here's an example of the text box code. Pay attention to the highlighted areas.

```
<&tbu2(0 B,0,23p,30p?p4,,,n,,(0.5,18,),(n),(,100),,n,,,,,1,2p1,,,,)>
```

2. The next box we want to think about is the 50% gray bar behind the “For Discussion Only” title. This does not need any relative placement or BL1 positioning because we know it will always sit in the same location. This picture box will need have the “Send to Back” *flag* applied.

```
<&pbu2(6 B,6,22p,52,,,k,n,(,18,),(n),(,100),,K,50,,m,,,,,,,"",,,)>
```

When this imports, it will come in with rounded corners on all 4 sides as seen here:



3. How we fix this is by bringing in a squared box with Black and 0% fill to knock out the gray which will give the custom shape.

```
<&pbu2(5.7 B,35,22p2,23,,,k,n,,(n),(,100),,K,0,,m,,,,,,,"",,,)>
```

The masking will take on this look. The dotted line around the box is for demonstration purposes only:



4. The order here is very important for this to work properly. We'll want the text box to be first, followed by the "Black 0% filled" picture box, and lastly have the gray 50% screened box that will be sent to the back. Here is how the code will appear:

```
<&tbu2(0 B,0,23p,30p?p4,,,n,,(.5,18,),(n),(,100)),n,,,,,1,2p1,,,,)>@ttl:Title
@h4:Text<&te><&pbu2(5.7 B,35,22p2,23,,,k,n,,(,n),(,100)),K,0,,m,
,,,,, "" ,,,)><&pbu2(6 B,6,22p,52,,,k,n,(,18,),(n),(,100)),K,50,,m,,,,
,,, "" ,,,)><&g(2,1,3)>
```

More Advanced Unsupported Shape Box

I will have to admit that this is tricky, but still achievable. When I first saw a similar example of this, I thought that the title bar and the perspective box below it would need to be made as an Adobe Illustrator image. After further study, I realized that this can be done entirely in QuarkXPress without the aid of outside art programs. Here is how this should import.

LEARNING PICTURE BOX TAGS

The purpose for working through all of these examples is to get the user more familiar with the different aspects of the program. It seems that a lot of people work with software like this for years, but fail to break new ground out of not having proper direction.

This is where the *corner type* comes into play to get the perspective effect on the title bar. Due to the complexity of this box, I have two choices on the number of boxes needed. I can have four boxes by keeping all of the type together, or I can reduce it to three boxes by keeping the title in the bar and breaking the text there. Since XML is the direction a lot of companies are going, it would probably be best to do with the text flow together by using four boxes. Here is how to achieve this:

1. The text box for the title is very basic with no special handling. The main issue is going to be the positioning.

108 Xtags Maximized

2. The 15% gray picture box behind the title needs no special handling outside of the screen and the frame around it.
3. The black perspective bar is where we run into our first real issue. This is done through Xtags using the “straight” *corner type* with a 1p6 corner radius. The only other special handling here is that it needs the “Send to Back” *flag*. Here is the Xtags string that makes up this box:

```
<&pbu2(0 B,21,24p,1p6,,,k,n,(,36,s),(n),(,100),,K,,,m,,,,,,,"",,,)>
```

This gives us a box that looks like this:



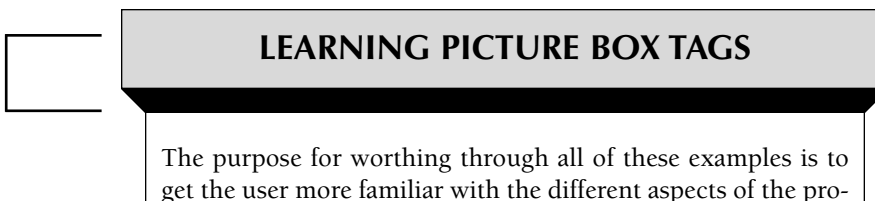
Since the box is sent to back and positioned with a 1p9 y coordinate, it will fall behind the title bar as shown below:



4. The picture box around the text area is a little complicated because it needs to be offset -6 TL1 and drop 29.4 points from the start of the text. It also includes relative placement with -2p5 depth reduction to accommodate the 2p5 spacing from the title to the stop of the rule. See example below. It also needs a “Send to Back” *flag* and a 0.5 point rule.

```
<&pbu2(-6 TL1,29.4,270,(35p?,R,-2p5,0),,,k,n,0.5,(n),
(,100),,K,0,,m,,,,,,,"",,,)>
```

2p5 space



5. The complete code for this is:

```
<&pbu2(0 B,0,24p,2p6,,,n,.5,(n),(,100),,K,15,,m,,,,,,,"",,,)><&pbu2(0 B,
21,24p,1p6,,,k,n,(,36,s),(n),(,100),,K,,,m,,,,,,,"",,,)><&tbu2(15 B,9.1,
21p6,35p?p3,,,n,,(n),(,100),,n,0,,,,,,,>>@ttl:Title
@tx:Complete text here<&te><&pbu2(-6 TL1,29.4,270,(35p?,R,-
2p5,0),,,k,n,0.5,(n),(,100),,K,0,,m,,,,,,,"",,,)><&g(4,3,2,1)>
```

Combining Elements

Some boxes have a lot of elements combined that make the floating object seem very difficult, but upon further investigation, it's more apparent that this can be handled without any problem. Take this example:

Box 3.3

Internet Home Businesses

With the onset of online shopping, auction companies, and the ease of use through wireless Internet, home business ventures are growing to be where some of the big money lies. Unlike the commercials that you see where it shows the entrepreneur sitting by his pool sipping a mixed drink, a home business does require a lot of extra time. A normal 40-hour work week can easily become a 65- to 70-hours a week job just to keep the business going strong. Once established, setting priorities and hours are easier, but still require a lot of push.

For such a small box, there will be six different strings using text, picture, and line tags. The bottom line could technically be put into the formats, but for demonstration purposes, I've left this as a line. Here's how we would start working with this box:

1. The text box is standard, so nothing has to be really thought out for this.
2. Since the text box needs no special handling, I will deal with the two boxes at the top. These are fairly simple with only shading that is needed. They will need to align next to each other. The width of the first box is at 5p6, so the second box needs to start at 5p6 x.

First 20% shaded box <&pbu2(0 B,0,5p6,2p4

Second 70% shaded box <&pbu2(5p6 B,0,18p6,2p4

3. The circle under the box number can be imported using a picture box tag with the oval "(o)" being indicated.
4. The line at the bottom will be brought in with using the line feature. The tag will be -11 BL1 from the text box with a 12 points y coordinate:

<&lbu(-11 BL1,12,0,24p,or,,1.5,,70,,,,)>

5. The circle gives you a chance to use the BR1 tag which isn't used as much as the others. I try to use this just to keep in practice. The picture box tag is reading -12p6 from the bottom right of the line to center the circle:

<&pbu2(-12p6 BR1,-6,12.5,12.5,,,,,(o),(n),(,100),,K,20,,m,,,,,,,"",,,)>

6. The combined tags for this box are as follows:

```
<&pbu2(0 B,0,5p6,2p4,,,n,,(n),(,100),,K,20,,m,,,,,"" ,,,)><&pbu2(5p6 B,
0,18p6,2p4,,,n,,(n),(,100),,K,70,,m,,,,,"" ,,,)><&pbu2(1p10 B,3,22,22,
,,, (o),(n),(,100),,,(1,1,1,1),m,,,,,"" ,,,)><&tbu2(11 B,8,22p2,35p?,,,
,n,,(n),(,100),,n,,,,,,>@bxt:Box Title
@bx:All of the box text.<&te><&lbu(-11 BL1,12,0,24p,or,,1.5,,70,,
,,)><&pbu2(-12p6 BR1,-6,12.5,12.5,,,,(o),(n),(,100),,K,20,,m,,,,
,, " " ,,,)><&g(3,2,1,4,5,6)>
```

7. If using a translation table, the simplified coding of this box will be:

```
[[b1]]@bxt:Box Title
@bx:All the box text[[b2]]
```

Side Margin Box with a Vertical Line

Side margin boxes should almost always be imported using Xtags, unless you are using Autopage and the box is very basic and you are using the margin note tags. Otherwise, the side elements should be constructed using the picture box, text box, or line tags. Here's an example of an element needing the text and picture box tags:

Building Skills

Gather in a group of two to three students. One strategy for becoming better at computer skills is to actually sit down and work on things you do not quite understand. For example, instead of going into a program and doing what you already know, dive into an area you are curious about and then see what you end up with.

The only real issue in this box is getting the rule to run the length of the text without having to adjust it. This is controlled by relative placement, but will need to have a depth reduction that includes the space from the title to the top of the rule and at the bottom of the text for the descenders.

We've covered this a little in other examples, but here's how that picture box will need to be written.

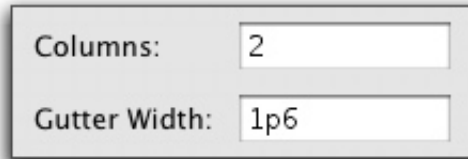
```
<&pbu2(0 TL1,15.5,1,(36p,R,-17.5,0),,,,n,,(n),(,100),,K,60,,m,,,,,"" ,,,)>
```

The complete tag will be written as:

```
<&tbu2(0 B,0,11p9,36p?p.2,,,n)>All of the text<&te><&pbu2(0 TL1,
15.5,1,(36p,R,-17.5,0),,,,n,,(n),(,100),,K,60, ,m,,,,,"" ,,,)><&g(2,1)>
```

Working with Multi-Column Text Boxes

When working with two- or three-columns, it's important to realize that the text will fill the first column, then if there's any text remaining, it will populate the second column. Xtags does not separate the columns into their own text boxes, but uses the columns and gutter for multi-columns:



The tag for two columns would be written as follows with the “2” for the number of columns and the 1p6 for the gutter width:

```
<&tbu2(0 B,0,26p,14p8?,,,n,,(n),(,100),,,,2,1p6,,,,,)>
```

Here's an example of how the text boxes would fill if there wasn't enough type to fill the 2-column text box.

<p>With the onset of online shopping, auction companies, and the ease of use through wireless Internet, home business ventures are growing to be where some of the big money lies. Unlike the commercials that you see where it shows the entrepreneur sitting by his pool sipping a mixed drink, a home business does require a lot of extra time. A normal 40-hour work week can easily become an endless 65- to 70-hours a week job</p>	<p>just to keep the business going strong. Once established, setting priorities and hours are easier, but still require a lot of push.</p>
--	--

It's very difficult to get a 2-column or 3-column layout where it will shrink-to-fit properly. I have tried different things like only using 1 column and then adding the supplementary columns later, but depending on how lines break, you still have to do some adjusting. I have found that this does come closer to the actual finished size, in most cases. There's really no perfect science when it comes to multi-column layouts.

Bringing it in as a single column is an option, or just realize that the columns are set correctly, and the biggest adjustment will be to the text and closing box depth. That is, if there isn't some decorative element at the bottom of the page.

Here's an example of how the text will pull in with a 2-column layout when there isn't enough text to fill the full depth of both columns. Even with the shrink-to-fit option, the text still fills the first column, then starts the next using the full depth.

Entertainment Problems and Solutions

Has the Creative Side in Hollywood Dwindled?

Problem Point 1

Creative movies are not the focal point anymore in Hollywood.

This used to be the case years ago when movies like Raiders of the Lost Ark, Star Wars, ET, etc. were being turned out.

Hollywood in 2004 is a grim situation. Studios are less interested in what will be remembered 10 years ago, and more concerned about which movies will be remembered for about 15 minutes. Hard working actresses like Catherine Mary Stewart and Cheryl Pollak rarely get starring roles because the MTV crowd would rather see Mandy Moore star in a movie because her recent video is on the #1 position on MTV's top 10 countdown.

Problem Point 2

The regurgated plots are constantly showing up everywhere with almost illegal stealing from other movies

Hollywood thinks people forget the plots of yesterday. Such is the case with *13 Going on 30*, which did well at the box office, but does this seem almost the same as *Big* starring Tom Hanks. If *Big* hadn't been a great movie, then it may be okay to use the same basic plot but substituting a female in the lead.

The code that makes up this box is reasonably simple with only some relative placement to deal with. The only real new thing here is the 2 columns and the gutter. Here is the full code:

```
<&pbu2(0,0,27p8,24.2,,,n,,(,n),(,100),,K,60,,m,,,,,,,"",,,)><&tbu2(10,5,
26p,7p9?,,,n,,(,n),(,100),,n)>@TTL:Title and subtitle<&te>
<&tbu2(0 BL1,1p,26p,25p?p.5,,,n,,(,n),(,100),,n,,2,1p3,,,,,,)>All of the
2-column box text here<&te><&pbu2(-10 TL1,-30,27p8,(40p,R,26,0),,,k,
n,,(,n),(,100),,K,10,,m,,,,,,,"",,,)><&g(1,2,3,4)>
```

The translation table would be a little more complicated here because we would need to break the text after the subtitle and start the two column text. The coded file would read:

```
@TTL:[[b1]]Title
@Subtl:Subtitle[[b2]]
@boxtxt:[[b3]]All of the 2-column box text[[b4]]
```

This is a little more complicated. Upon finishing this box, the adjustment would just be adjusting the 10% gray box and the 2-column text.

Inline Text Elements

Let's break away from boxes for a little bit. I've focused a lot on these and they are really what the bulk of this chapter is all about, however, I feel that I'd be doing you a disservice if I only showed you what is possible there. I want to touch on text elements that can save time when working in the chapters.

If using Autopage, most text elements will be in the library to be extracted during the AutoTag conversion as a side art underlay, but I find that these can be imported as easily as pulled from a library. Let's look at this H1 head below:

THE ROLE OF MARKETING ONLINE

As mentioned in the previous paragraph, using this as an Autopage side art underlay is probably the best way and is technically a correct answer.

I understand that not every reader owns Autopage, so for those users I would say there is an easy way that can happen during the coding instead of rooting around for it in the library. This is done by grouping this together and importing it onto the page. Here is how we are going to accomplish this:

1. We need to create the element. When you first look at it, you probably feel that the rule should be in the style, but I decided to make it all one grouped element with a vertical and horizontal rule and a 30% black picture box. That way there's less chance of having the trapping off.

2. With Xtags, the tagging will be:

```
<&lbu(298,15,180,294,or,i,0.5,,,1,,)><&lbu(4,7.6,270,7.6,or,i,0.5,,,1,,
,><&pbu2(0,0,8,8,,,,,(n),(,100),,K,30,,m,,,,,"",,)><&g(3,2,1)>
```

3. I imported this grouping on the top of the page opposed to the paste-board. This way it will actually be sitting visibly on the page. I cannot get the coding to do this exactly with Xtags, but manually I can get this group of boxes to anchor with the head, so I will go through and anchor all of these with the H1 heads. To do this:
 - a. Copy the grouping.
 - b. Go to the H1 head and return immediately afterwards.
 - c. Create a style sheet called H1_Bar based on the H1 head. This style will need a 0 left indent (instead of 1p3) and leading needs to be 3pts.
 - d. Paste the grouped item into the text flow.
4. Now all of these items will float with the H1 head while paging. If I can only figure out how to get Xtags to allow me to code these into the text, I'll be extremely happy, but this will save time during the paging process.

Table with Art Handling

The following example is a table (still treated like a box), but with a decorative piece of art that comes in at the bottom left corner of the page.

Table 13.1 Things to look for when using picture box tags

- Try to use as few boxes as possible during the set up. If I can use insets or First Baseline to accomplish this, I'll spend a little extra time to keep the boxes as user-friendly as possible.
- Make sure if setting up for XMLxt usage that the text can read through without any text breaks; otherwise, the XML will not stay well-formed. If the box isn't being used with XMLxt, this isn't an issue.
- If there are any Quark blends, I set them as Adobe Photoshop files because Xtags doesn't currently support blends.
- Always have the box built perfectly with trapping, alignment, and so on before writing the Xtags strings to ensure everything will work correctly when importing the text.

This box would look much better in color, but for demonstration purposes we'll just use the different shades of gray. The approach I would take first on this is by looking at what requires any relative placement or TL1, BL1, etc. The three obvious boxes are the top 70% bar, the 25% bar around the table title, and the text area. There will need to be a slight indent on the positioning on the pasteboard to accommodate the piece of art in the bottom left corner. If not enough white space is available on the pasteboard, an error will occur.

The next steps are as follows:

1. Make the 15% background behind the text -13.5 TL1 and starting at 1p y from the top of the text area. Shrink-to-fit the depth using relative placement, but reduce 3 points from the total depth. This also needs to have the "Send to Back" flag.

```
<&pbu2(-13.5 TL1,12,26p5,(30p,R,-3,0),,,k,n,,(n),
(,100),,K,15,,m,,,,,,,"",,,)>
```

2. The piece of art in the bottom left corner will need to be -3pts BL1, with a -8p1 y. No matter how deep the text is, the art will always place in this position.

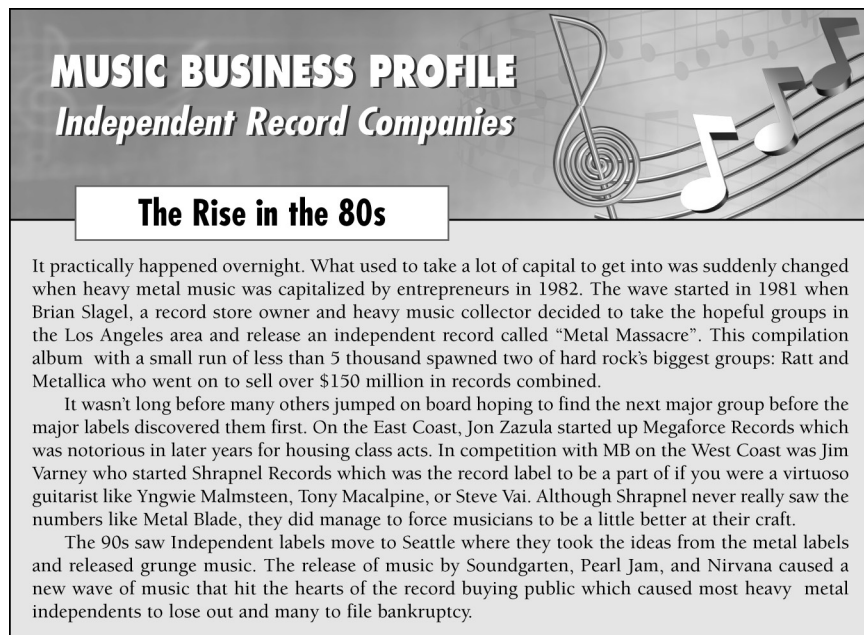
```
<&pbu2(-3 BL1,-8p1,100,100,,,k,n,,(n),(,100),,,,m,,,,,"56 GB
Disk:Xtags Book:Images:Chapter 05:ch05_008.eps",,,)>
```

3. The complete code for this box will be:

```
<&pbu2(9 B,0,26p5,24,,,n,,(n),(,100),,K,70,,m,,,,,"",,,)><&pbu2(17 B,
2.5,5p,24,,,n,,(n),(,100),,K,25,,m,,,,,"",,,)><&tbu2(5.5 TL1,6.5, 24p6,
30p?p.5,,,n,,(n),(,100),,n,,,,,,,)>All the box text<&te>
<&pbu2(-13.5 TL1,12,26p5,(30p,R,-3,0),,,k,n,,(n),(,100),,K,15,,m,,,
,,,,"",,,)><&pbu2(-3 BL1,-8p1,100.876,100.546,,,k,n,,(n),(,100),,,,m,
,,,,,"56 GB Disk:Images:Chapter 05:ch05_008.eps",,,)><&g(5,4,3,2,1)>
```

Complicated Box with Art

Many would argue that this box should be set up using the T=E in Autopage, and I would do this if using an XML workflow. If this box is set up correctly, this will perfectly import every time. I'm going to break down how these strings can be reduced to a very simple translation table entry. This may look a little overwhelming, but I left some of the default information in the fields. Look carefully at the tags; there's a lot going on.



If you've been following along in all of the other examples, you should be ready to tackle this. Xtags sometimes seems like learning Algebra in a way. If you get your foundation down, there isn't much you can't accomplish with it. This box does have a lot going on. The main concerns are the shadow on the text, the art in the background, and trapping the rule around the entire box. This is accomplished by:

1. The image box will come in with a 1pt rule around it.

116 Xtags Maximized

2. The next step is to get the Super Title and the Title to come in with the shadow behind it. This is accomplished by just having a .8 x offset and a .9 y offset and duplicating the title. Here's an example of the code:

```
<&tbu2(1p8.2,2p1.9,22p5,5p5 and <&tbu2(1p7.6,2p1,22p5,5p5
```

3. The "Rise in the 80s" subtitle box is brought in with "center" on the Vertical Alignment and "1p7" for the Baseline Offset.

```
<&tbu2(3p1,8p4,17p7,2p8,,,n,1,(K,n),(60,100),"Solid",W,
100,,1,,,1p7,,c,,,"")>
```

4. The part that can make or break this is getting the rule at the top of the text area to appear seamless from the bottom rule on the art box. This is accomplished by trapping the art 1pt. under the other box, and sending it to the back. This box starts on the 9p11 y coordinate, where the top box has a total depth of 10p. This will underlay the other box.

```
<&tbu2(0,9p11,40p6,22p?,,,k,n,1,(K,n),(,100),,K,10,,1,
,(21,12,8,12),,,t,,,"")>
```

5. The code that makes up this box is as follows:

```
<&pbu2(0,0,40p6,10p,,,n,1,(K,n),(100,100),"Solid",K,0,,m,,,1,-1,,,56
GB Disk:Art:Photos:jd01.tif",,"")>
<&tbu2(1p8.2,2p1.9,22p5,5p5,,,n,,(K,n),(,100),,n,100,1,,,,t,,,"")>
@SMP_BX1_HD_S:Music Business Profile
@SMP_BX1_SHD_S:Independent Record Companies<&te>
<&tbu2(1p7.6,2p1,22p5,5p5,,,n,,(K,n),(,100),,n,,1,,,,t,,,"")>
@SMP_BX1_HD:Music Business Profile
@SMP_BX1_SHD:Independent Record Companies<&te>
<&tbu2(3p1,8p4,17p7,2p8,,,n,1,(K,n),(60,100),"Solid",W,100,,1,,,19,c,,
,"")>@SMP_BX1_SSHD:The Rise in the 80s<&te>
<&tbu2(0,9p11,40p6,22p?,,,k,n,1,(K,n),(,100),,K,10,,1,,(21,12,8,12),,,t,,
,"")>@SMP_BX1_F:It practically happened overnight<&te><&g(1,2,3,4,5)>
```

This would be broken down in the translation table as:

```
[[bb1]] <&pbu2(0,0,40p6,10p,,,n,1,(K,n),(100,100),"Solid",K,0,,m,,,1,-1,,,56 GB Disk:Art:Photos:jd01.tif",,"")> <&tbu2(1p8.2,
2p1.9,22p5,5p5,,,n,,(K,n),(,100),,n,100,1,,,,t,,,"")>
[[bb2]] <&te><&tbu2(1p7.6,2p1,22p5,5p5,,,n,,(K,n),(,100),,n,
100,,1,,,,t,,,"")>
[[bb3]] <&te><&tbu2(3p1,8p4,17p7,2p8,,,n,1,(K,n),(60,100),
"Solid",W,100,,1,,19,c,,,"")>
[[bb4]] <&te><&tbu2(0,9p11.05,40p6,22p?,,,k,n,1,(K,n),(,100)
,,K,10,,1,,(21,12,8,12),,,t,,,"")>
[[bb5]] <&te><&g(1,2,3,4,5)>
```

The text file would look like this:

```
@SMP_BX1_HD_S:[bb1]Music Business Profile
@SMP_BX1_SHD_S:Independent Record
Companies[[bb2]]@SMP_BX1_HD:Music Business Profile
@SMP_BX1_SHD:Independent Record
Companies[[bb3]]@SMP_BX1_SSHD:The Rise in the
80[[bb4]]@SMP_BX1_F:It practically happened overnight....[[bb5]]
```

In the input file, you'll want the macros ([bb1], [bb2], etc.) to run in. If not, your document will have a runaway space at the end of each text box. Writing a detailed coded sample reduces the complexity for everyone involved. I personally write a MacPerl script inserting the tags for these boxes. For example:

```
s/(\@SMP_BX1_HD_S:)([A-Z])/$1\[bb1]$2/g;
```

This pattern replacement places the [bb1] code between the style name and the first character of text.

```
s/(\@SMP_BX1_SMP_SHD_S:)(.+?)(\n)/$1$2\[bb2]/g;
```

This puts the [bb2] at the last character of type and removes the return so the text will run in with the next style.

MacPerl or AppleScript can keep the control in your hands rather than a markup department. My advice is to follow whatever works best for you. If you begin scripting you will quickly realize that productivity will increase.

As you have seen earlier in the book, I will also use AppleScript to accomplish this as well. It all depends on the project. With OSX, I've found that I enjoy using AppleScript more to accomplish tasks like this. It also helps because you can put error checking in to reduce careless coding errors.

Complex Box with Seven Xtags Strings

This is another box that I would do with Xtags rather than have to adjust manually along the way. It is complicated, yet when you break down the macros and the translation table, it all makes sense.

Note: The 8 pt. decorative box at the bottom corner will always snap into this location due to the usage of shrink-to-fit capabilities set in the other boxes. There's also BR1 usage applied for this effect. The shading behind the "Problem Points" is set in the style sheet rather than expecting Xtags to insert these. This takes some of the complication out of this.

There is also slight overlapping on a couple of the elements. For example, I have the [hw1] *Sent to Back* so it will trap behind the [hw2] "Hollywood Clichés," which starts at 14p, to overlap the other box by .5 pts. I believe in trapping as much as possible on a box like this.

HOLLYWOOD CLICHÉS	THE HOLLYWOOD WHEEL
Identifying the Problems and Creating a Solution	
PROBLEM POINT 1	<p>CREATIVE MOVIES ARE NOT THE FOCAL POINT ANYMORE IN HOLLYWOOD. THIS USED TO BE THE CASE YEARS AGO WHEN MOVIES LIKE RAIDERS OF THE LOST ARK, STAR WARS, ET, etc. were being turned out. Even horror movies like <i>Halloween</i> had a unique twist.</p> <ul style="list-style-type: none"> Hollywood in 2004 is a grim situation. Studios are less interested in what will be remembered 10 years ago, and more concerned about which movies will be remembered for about 15 minutes. Hard working actresses like Catherine Mary Stewart and Cheryl Pollak rarely get starring roles because the MTV crowd would rather see Mandy Moore star in a movie because her recent video is on the #1 position on MTVs top 10 countdown.
PROBLEM POINT 2	<p>THE REGURGATED PLOTS ARE CONSTANTLY SHOWING UP WITH ALMOST ILLEGAL STEALING FROM OTHER MOVIES</p> <ul style="list-style-type: none"> Hollywood thinks people forget the plots of yesterday. Just because hip hop artists can take a song like "Dream On" and spin a new twist on it, doesn't make us forget that it's a deliberate steal. Such is the case with this years <i>13 Going on 30</i> which did well at the box office, but does this seem almost the same as <i>Big</i> starring Tom Hanks. If <i>Big</i> hadn't been a great movie, then it may be okay to use the same basic plot but substituting a female in the lead isn't really pulling anything over anyone's eyes.
<p>STOP GOING TO THE MOVIES Maybe it's time people quit going to see movies that they know aren't worth seeing. If we've already seen it, don't go to the theater to see it. Wait until it comes on video. This will send a strong message to the studios that if they want to deliver garbage, it must go straight to video.</p>	

How the coded sample and the translation table need to be set is shown below. To have this become second nature, my suggestion is to practice creating functional boxes whenever time allows.

Coded File

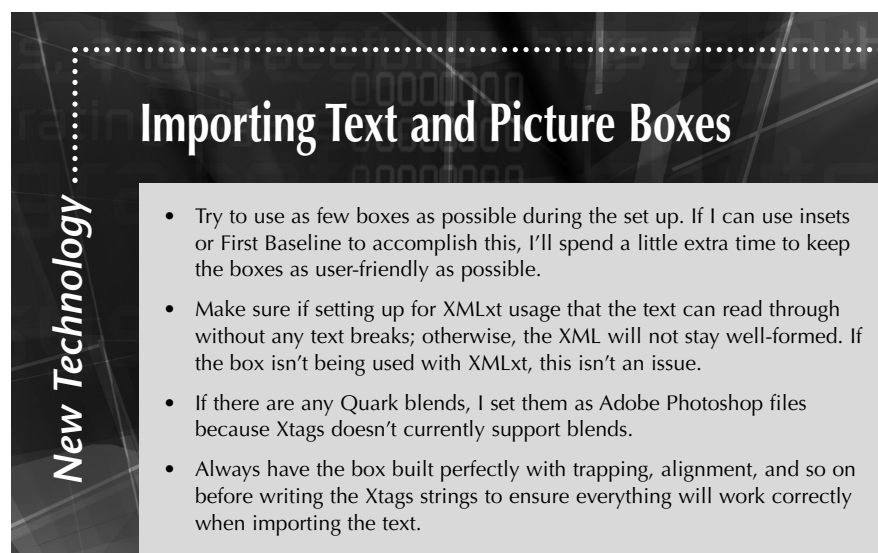
```
[[hw1]]@BOX_STTL:Hollywood Clichés
[[hw2]]@BOX_TTL:The Hollywood Wheel
[[hw3]]@PP:<@PP_TTL>Problem Point #1<@$p><\i> Creative movies are...
[[hw4]]@BOX_COL:Identifying the Problems and Creating a Solution[[hw5]]
```

Translation Table

```
[[hw1]] <&tbu2(0, 0, 14p,1p4,,,n,,,,,"S82" ,,,,,8,,):>
[[hw2]] <&te><&tbu2(13p11.5, 0, 28p6.5,1p4,,,k,n,,,,,"S20" ,,,,,11.5,,):>
[[hw3]] <&pbu2(13p11.5,1p2,28p6.25,p2,,,n,,,,,"S82" ,,,,,,
,)><&tbu2(0p, 3p2.8,42p6,56p?+7,,,k,n,,,,,"S10" ,,, ,,,18.8,
,):><&te>
[[hw4]] <&te><&pbu2(0p BL1,-p.5,42p6,p2,,,n,,,,,"S82" ,,,,,,)>
<&pbu2(-8 BR1,-8,p8,p8,,,n,,,,,1,"S82" ,,,m,,,,,)><&tbu2(0p, 2p,
42p6,1p5,,,n,1,"S82" ,,,K,0,,,,,11,, ,):>
[[hw5]] <&te><&g(1,2,3,4,5,6,7)>
```

Complicated Box with Rotated Text

Rotated text is something we really haven't touched on much yet, but is easier than most of the other concepts in this box. The other issues in this box have been dealt with in previous examples. That doesn't mean that there won't be a few difficulties setting this up, but it should be easier than when the chapter first began. Here is how the box should import:



Let's look at what we're dealing with here. There is a large piece of art in the background, two dotted lines, rotated text, and a text box requiring relative placement. When I approach a box, I don't try to figure it all out the second I get into it, but I do make note of anything troublesome. Overall, this box isn't too complicated, but there are a few difficult scenarios. Let's begin by following these steps:

1. My first step, normally, would be to bring in the art background. However, how do we know how deep it will be. This will not be known until after the text comes in and we can put relative placement on it. This is the difficult part of the box.
2. Instead of dealing with the image, I will start positioning what will be standard. This will be the rotated super title, the title, the two dotted lines, and the text box.
3. The first box I am going to bring in is the text box with the 15% shading and the multiple inset. I am bringing this in at a 9p x positioning on the pasteboard. This gives the rotated text room to rotate:

```
<&tbu2(9p B,5p4,23p6,35p?,,,n,(n),(,100),,K,15,,, (9,9,9,7),,,,,)>
```

120 *Xtags Maximized*

4. Due to the text box being positioned, I need to calculate where the background art will position. This requires a relative placement and will be positioned -4p TL1 with a -5p4 y coordinate because the 15% gray text box begins on 4p x and 5p4 y coordinates. The depth will be relative with an additional 5p4. This box also needs to have the “Send to Back” flag.

```
<&pbu2(-4p TL1,-5p4,27p6,(35p?,R,5p4,0),,,k,n,,(n),(,100),,n,0,,m,,,
,,,,"56 GB Disk:Xtags Book:Images:Chapter 05:ch05_009.eps",,,)>
```

The piece of art will need to be created for the full page depth in case the text box takes up the majority of the page depth. To avoid running into problems later, don't make the art only half the page size.

5. The rotated text box with the “New Technology” title will need “90” for the degrees in the rotation field (#5). You may have to play a little with the x and y coordinates to get this to position where you want it.

```
<&tbu2(24.5 B,10p,10p9,27,90,,,n,,(n),(,100),,n,,,,,,,,,>
```

6. The last step is getting the boxes and lines to fall in their proper positioning. The lines are important to make sure they intersect properly. It is also important to note that you can use “4” to bring in the dotted line or the “All Dots” definition that needs to be within quotations.

Also, in the color, make sure that the (W,n) is on so the dotted rule doesn't pick up the gap color as white also. This will give the appearance of a straight line if this isn't specified.

```
<&lbu(7p B,1p2,0,306,or,,2,(W,n),,4,,,)>
```

```
<&lbu(7p B,1p2,-90,49,or,,2,(W,n),,"All Dots",,,,)>
```

7. The complete code is as follows:

```
<&tbu2(9p B,5p4,23p6,35p?,,,,n,,(n),(,100),,K,15,,, (9,9,9,7),,,,
,,)>Text box material<&te><&pbu2(-4p TL1,-5p4,27p6,(35p?,R,5p4,0),
,,k,n,,(n),(,100),,n,0,,m,,,,,,,,,"56 GB Disk:Xtags Book:Images:Chapter
05:ch05_009.eps",,,)><&te><&tbu2(9p B,2p9,19p6,2p2?p.5,,,n,,(n),
(,100),,n,,,,,,,,,>Title<&te><&tbu2(24.5 B,10p,10p9,27,90,,,n,,(n),
(,100),,n,,,,,,,,,>New Technology<&te><&lbu(7p B,1p2,0,306,or,
,2,(W,n),,4,,,)><&lbu(7p B,1p2,-90,49,or,,2,(W,n),,"All Dots",,,,)>
<&g(1,2,3,4,5,6)>
```

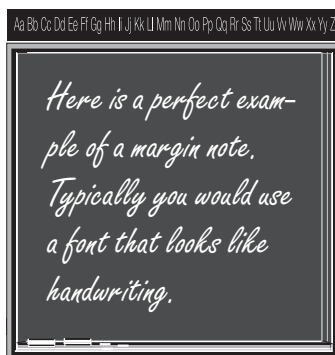
8. When using a translation table, the coded file would be handed as:

```
[[b1]]Text box material[[b2]]Title[[b3]]New Technology[[b4]]
```

Overall this box is somewhat tricky, but it doesn't need to impact the markup department once the codes are set into a translation table and you mark up the coded sample.

Side Margin Box Expandable

I've seen pagers make a major mistake when working with side margin boxes or regular boxes that need to be created as art. Especially in a case like this where the bottom of the box is distinct from the rest of the box. Look at this example:

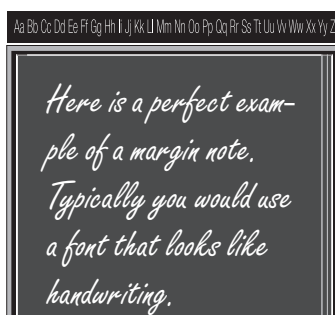


As you can see, the bottom of the box is unique to the rest of the element. I've actually seen pagers have an art department create 10 different sized boxes that would be treated like this. They have the art named "margin_10p.eps", "margin_11p6.eps", "margin_13p.eps", etc. Handling it this way creates a huge mess for everyone. During edits, you are scrambling to put in the correctly sized piece. The more effective way is to have one large piece that can handle any depth of the margin note. Then have Xtags import as three pieces using relative placement for positioning. Here's how we will accomplish this:

1. The first objective is to bring in the text box. This is just standard text box handling with a slight pasteboard offset and a 2p4 y coordinate.
2. Once the text depth is established, we'll need to bring the first part of the art element relative to the depth of the text plus 2 points with the box positioning to the top left minus the x and y coordinates of the text box start.

```
<&pbu2(-15 TL1,-2p4,10p6,(25p,R,30,0),,,k,n,,(,n),(,100),,n,,,m,,,
,,, "56 GB Disk:Xtags Book:Chalkboard.eps" ,,,)>
```

This imports this amount of the image with the text:



122 Xtags Maximized

3. We'll then want the bottom of the box to close this off. The way to do that is by having the end of the box in a 1p6 depth box that only contains the bottom of the element. This will be offset by 42p3.5 and positioned to the bottom left of the top portion of the art:

```
<&pbu2(0 BL1,-1,10p6,1p6,,,n,,(n),(,100),,n,,,m,,,0,-42p3.5,,,56 GB
Disk:Xtags Book:Images:Chapter 05:Chalkboard.eps" ,,) ><&g(3,2,1)>
```

This string brings in this small piece of art to close off the bottom of the margin note:



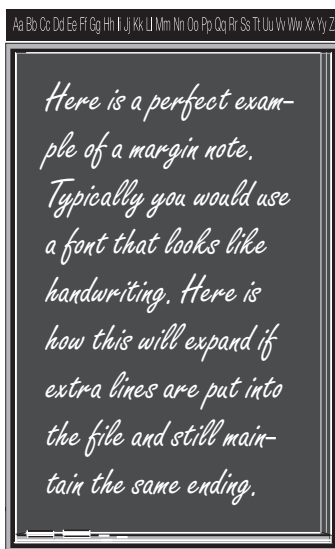
4. The entire tag put together will then reflect the following:

```
<&tbu2(15 B,28,8p,25p?,,,n,,(n),(,100),,n,,,,,,,) >@mn:margin note
text here.<&te><&pbu2(-15 TL1,-28,10p6,(25p,R,30,0),,k,n,,(n),
(,100),,n,,,m,,,,,"56 GB Disk:Xtags:Chalkboard.eps" ,,) ><&pbu2(0 BL1,
-1,10p6,1p6,,,n,,(n),(,100),,n,,,m,,,0,-507.5,,,56 GB Disk:Xtags
Book:Images:Chapter 05:Chalkboard.eps" ,,) ><&g(3,2,1)>
```

5. By handling it this way, the tags can always be set up the same for the markup department using:

```
@mn1:[mn]margin note text here[mn2]
```

6. This shows how the margin note can expand, but still close the same allowing for any extra text without having multiple common elements to have to position each time. This proves the time saving power of Xtags.



Picture Box Using Skew

Over the years that I have been working with Xtags, I haven't found many uses for the #6 *Boxskew* field. The reason for this is usually the designers create the images the way they want them to be used in the design without using the skew override. I have seen many designs where the skew could be used and it really depends whether the designer sets the skew in the image or in Quark. Even though my current workflow does not use it that often, I still can see where there are a lot of uses for this. A good example is in this box example shown here:

Innovative Technique

The purpose for working through all of these examples is to get the user more familiar with the different aspects of the program. It seems that a lot of people work with software like this for years, but fail to break new ground out of not having proper direction. The goal is to show as many different examples as possible covering all of the picture and text box options through the examples.

The box has a simulated lowercase “i” that is built easier by using *Boxskew* than by finding an actual font for this. To do this:

1. Take a 20% filled picture box for the top box and add a -15% skew to it. This will give the following result:



2. The bottom picture box will need the same treatment, but will need to have more depth. This will also need a runaround with 9 pts on the right side. The two boxes together will have this coding:

```
<&pbu2(0,0,42,16,-15,,(n),(100),,K,20,,m,,,,"",,,)>
<&pbu2(15,21,45,84,-15,,(n),(100),,K,20,(1,1,1,9),m,,,,,"",,,)>
```

3. The other notable is that the text area is not skewed. It is merely playing off of the runaround giving it that effect. It is important that the text box has the “Send to Back” *flag*.
4. The complete code for this box is:

```
<&pbu2(0,0,42,16.277,-15,,(n),(100),,K,13,,m,,,,,"",,,)>
<&pbu2(15,21,45,84,-15,,(n),(100),,K,20,(1,1,1,9),m,,,,,"",,,)>
<&tbu2(27,10.8,271,25.5,,(n),(100),,n,,,,,,)>@title:Innovative
Technique<&te><&tbu2(49,38,23p,15p?,,,k,n,,(n),(100),,n,,,,,,
,,,)>All the text here<&te><&g(1,2,3,4)>
```

Using Lines for Top and Bottom of Box

This is an example of a picture box requiring lines to be at the top and the bottom of the screen. This appears easy on the surface where the pager could probably include the lines using the rules in the formats, but this is one of those situations where you have to look close at the example.

The purpose for working through all of these examples is to get the user more familiar with the different aspects of the program. It seems that a lot of people work with software like this for years, but fail to break new ground out of not having proper direction.

The line at the top and the bottom has a 0% Black “Gap” to give the “White” effect between the two lines. If you were using a rule above in the formats, the “Gap” is not available, so it will end up with this effect instead:

The purpose for working through all of these examples is to get the user more familiar with the different aspects of the

Notice how the gray is showing through the rules. That is why we need to use the “unanchored line tags” instead so this will be white. To do this properly:

1. This is very basic. Start the box with the unanchored line followed by the screened unanchored text box. The text box will need 5 points additional depth added with an 18 points first baseline:

```
<&tbu2(0,0.5,174,150?p5,,,k,n,,(n),(,100),,K,15,,,,,18,,,,)>
```

2. The bottom line will need to overlay the screen by -.5 point so it can trap properly.
3. The complete tags are:

```
<&lbu(0,0,0,174,or,,4,(,K),(,0),"Double",,,,)><&tbu2(0,0.5,174,150?p5,,,k,n,,(n),(,100),,K,15,,,,,18,,,,)>All text goes here<&te><&lbu(0 BL1,-.5,0,174,or,,4,(,K),(,0),"Double",,,,)><&g(3,2,1)>
```

4. The markup department will need to only have two tags to make this work when using a translation table:

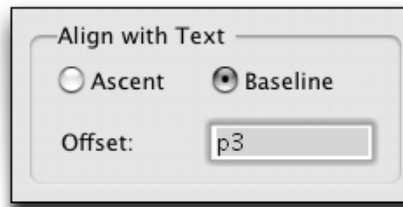
```
[[b1]]@bx1:All text[[b2]]
```

MathType Anchored with Xtags

Many pagers prefer to have their equations anchored in the text flow rather than as floating objects. When anchoring a MathType equation, do not use the *Anchor* setting: *Ascent*. *Ascent* will cause type to run up alongside the equation unless you put a large amount of space below the equation in the format. Depending on the depth of the equation, you'll be manually adjusting most of the equations. *Ascent* would only be used if you had any equations flowing within a paragraph.

If you have equations that sit on their own lines with space above and below, Xtags can handle this without any problem. Here is where to begin:

1. Anchor an equation into the text flow. Set this equation 4p6 w. by 2p6 h. to follow along with this exercise (normally it can be any size). It's critical to know the necessary white space above and below the equation. For this example, I will need 6 points of white space above and below the equation. Go to *Item:Modify* in the Quark pull down menus. In the *Align with Text* section, select "Baseline", and put the "Offset" to p3.



There's generally around 3 points of white space padded in a buildup MathType equation, so this will make sure my base-to-base space stays in the proper range. I always let my equations flex at least a point, so you'll still be within the specs.

2. Set your space above in the *Paragraph Attributes* of the "Formats" to 6 points. Do not use a hard return above and below the equation. Next, center the equation, making sure there are no indents. Set your leading to "auto" (This is the only time I would suggest "auto"). Your format window should be set as follows:
3. Update your equation spec to reflect these settings in the *Paragraph Attributes*. I have refrained from using an Autopage H&J in this because I feel there will be enough flexibility with the varying sizes of the equations. If you do use one, use it sparingly for variability or the pages could look unbalanced.
4. Select the equation and change the "Runaround" values to 0. Because this is anchored, you'll have to have at least this value.

126 *Xtags Maximized*

5. Xtags is necessary to automate this process. Don't panic—the most time consuming part is getting the Quark formats and anchored info correct. Then go to: *Edit:Copy XTags Text*.
6. Go to the pasteboard, draw a text box, and *Paste*. You will now have a code that looks like this:

```
<v2.00><e0>
@DM:<&pb(54,30,(b,3),0,(K,n),(100,100),"Solid",W,100,(0,0,0,0)
,m,100,100,,,,,"56 GB Disk:AP:Equations:EQ_005.eps" ,,"")>
```

Depending on the width and depth of your equation, you'll have different values. I've highlighted the values that need to change.

7. Change the width and height values to be: **26p?,9p?**. If you have a 30p text width, this can be 29p?. I usually bring this in 1 pica from the total text width.
8. The pathname to your art is going to be different, but this shouldn't change from what you have during the *Copy Xtags Text* stage.
9. By eliminating unnecessary Xtags information, your path resemble this:

```
@DM:<&pb(26p?,9p?,(b,3),,(K,n),(100,100),,W,100,(0,1,0,1),m,10
0,100,,,,,"56 GB Disk:AP:Equations:EQ_004.eps" ,,"")>
```

10. You'll need to cut out part of this and paste it into your Xtags translation table and label as shown here:

```
[[dm1]] <&pb(26p?,9p?,(b,3),0,(K,n),(100,100),"Solid",W,
100,(0,1,0,1), m,100,100,0,0,0,0,"Your folder pathname:
[[dm2]] ",,"")>
```

You'll see there are two individual parts. The first part makes the box and finds the art, the second part closes it off.

11. The input/coding file should look similar to the following when set up:

```
<&tt2"yourtitle.ttl">
@T1:The following equation is needed:
@DM:[[dm1]]EQ_001.eps[[dm2]]
@T1:This is the text following.
```

When the markup department types the [[dm1]]equation path name[[dm2]], it will automatically bring in each equation where needed if named properly. This can also be achieved using MacPerl or writing an AppleScript to number each new EQ number as it comes to it.

12. You are ready to import the equations into your document. I always test a couple first to make sure they are working.

Oval Text Box with Offset Shadow

When the content is placed into a rounded box with a shadow, this is typically positioned inline and used as a “Tip” or possibly a “Quote”. You may even see this as a margin feature, but only smaller in width. This is fairly easy to do, but requires that you have certain spacing built into both boxes to achieve the result.

TIP

Database programs like FileMaker Pro can take scripting and data entry and combine the two making one of the most cost-effective programs to work out of.

You will need to use the “oval” box type, shade the background 15%, add a *First Baseline* of 2p, and add 1p6 to the box depth field (9 points in the code).

```
<&tbu2(0 B,0,25p,18p?p9,,,n,(o),(n),(,100),,K,15,,,,,2p,,,,,>
```

The shadow is fairly easy, but you will want it to position properly so it only picks up the amount shown above to give an almost 3 dimensional effect. For this to happen, we’ll need to offset the picture box or text box (your choice) top left with the x coordinate by 3pts, the y coordinate by 4pts, and using the “oval” box type again. The rest of the box requires relative placement, an additional 3 points to the depth, a 45% shade, and the “Send to Back” flag.

```
<&tbu2(3 TL1,4,25p,(22p,R,3,0),,k,n,(o),(n),(,100),,K,45,,,,,,,>
```

The more text that is contained in the box will change the appearance to become more of a circle than the oval shape seen above. Typically to maintain the oval look, the designer would need this to fit content that was no more than 7 or 8 lines to maintain some uniformity throughout the design.

Here is the code that makes up the box:

```
<&tbu2(0 B,0,25p,22p?p9,,,n,(o),(n),(,100),,K,15,,,,,24,,,,,>
All text here<&te><&tbu2(3 TL1,4,300,(22p,R,3,0),,k,n,(o),
(n),(,100),,K,45,,,,,,,><&te><&g(2,1)>
```

The translation table entries would be:

```
[[bx1]] <&tbu2(0 B,0,25p,22p?p9,,,n,(o),(n),(,100),,K,15,,,,,24,,,,,>
[[bx2]] <&te><&tbu2(3 TL1,4,300,(22p,R,3,0),,k,n,(o),(n),(,100),
,K,45,,,,,,,><&te><&g(2,1)>
```

Complicated Box with Multiple Elements

Even if you don't plan on setting up a box like this, I still think this one would be good practice for you. The whole idea of this book is that I want others to not feel limited when they have a pager walk up and ask, "Can this be set up in Xtags?" I would hope your response in the future will be, "Yes. Give me a few minutes to figure it out."

I thought I would throw in this box because it covers so many different elements all in one. It looks almost easier to do this as art or out of a library, but I thought by combining it together for Xtags to bring in, that you might see how you aren't limited at all. Here's how the final box should appear:

Review Box 12.1

- Try to use as few boxes as possible during the set up. If I can use insets or First Baseline to accomplish this, I'll spend a little extra time to keep the boxes as user-friendly as possible.
- Make sure if setting up for XMLxt usage that the text can read through without any text breaks; otherwise, the XML will not stay well-formed. If the box isn't being used with XMLxt, this isn't an issue.
- If there are any Quark blends, I set them as Adobe Photoshop files because Xtags doesn't currently support blends.
- Always have the box built perfectly with trapping, alignment, and so on before writing the Xtags strings to ensure everything will work correctly when importing the text.

The following elements are here: two rectangular picture boxes, a curved picture box, a rotated picture box, two text boxes, a line, and a picture box resembling a 1pt rule. That is a lot of material to get put together, but it is not as difficult as you might think. How I start this box is with the rounded corner box (I have these 50% shaded for demonstration purposes) that will come in like:



Following this, I will add a 4p square box to start at the top left edge of the first box. And I will also bring in a rotated 45° box over the top of the two boxes to give the triangle cut into the side. The long rectangle just overlaps the first box. The first box needs to have the "Send to Back" flag.

```
<&pbu2(10.6 B,0,11p6,2p6,,k,n,(18),(n),(100),K,,,m,,,,,
,,",,,)><&pbu2(10.6 B,0,4p,2p6,,,n,,(n),(100),K,,,m,,,,,"",,,)>
<&pbu2(0 B,4.5,21.15,21.15,45,,,n,,(n),(100),K,0,,m,,,,,"",,,)>
<&pbu2(84.6 B,18,273,12,,,n,,(n),(100),K,,,m,,,,,"",,,)>
```


For this example, I put shading on the boxes and around the rule of the rotated box for demonstration purposes. The four boxes together will appear as:



As you can see from the code, none of the top bars have any top left or bottom left placement. They will come into this position exactly.

Next we need to deal with the line that is running below the entire grouping. This will need to read off the bottom left of the first box that was brought in. This would read off of the first box in the code as shown here:



Because this box is used for positioning, we will need to use a “BL5” tag because there will be four picture boxes and one text box imported prior to this appearing. The coding for the line will be:

```
<&lbu(0 BL5,3,0,347,or,,1,,,,,>
```

The main text area will follow the rule with a 1p3 bottom left offset and a 9 point y offset below the line. The left line running the length of the text will be brought in as a picture box with a relative placement that will be 7 points deeper to accommodate the offset, but not the text descenders:

```
<&tbu2(1p3 BL1,9,27p3,25p?,,,,,(n),(,100),,,(1,1,1,1),,,
,,,,,>Text here<&te><&pbu2(-1p3 TL1,-
9,1,(25p,R,7,0),,,n,,(n),(,100),,K,,,m,,,,, " " ,,>
```

The entire code is as follows:

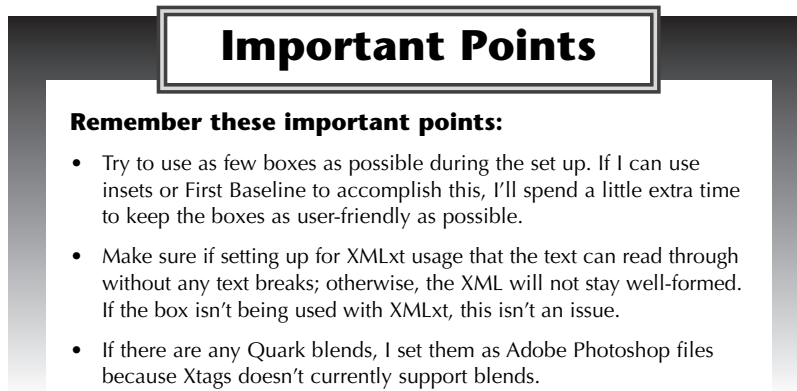
```
<&pbu2(10.6 B,0,138,30,,,k,n,(,18),(,n),(,100),,K,,,m,,,, " " ,
,><&pbu2(10.6 B,0,48,30,,,n,,(n),(,100),,K,,,m,,,,, " " ,>
<&pbu2(0 B,4.5,21.15,21.15,45,,,n,,(n),(,100),,K,0,,m,,,
,,, " " ,><&pbu2(84.6 B,18,273,12,, ,n,,(n),(,100),,K,,,m,,,
,,, " " ,><&tbu2(29.6 B,8,108,16,,,,,(n),(,100),,n,,,,,
,,>@RB:Review Box 12.1<&te><&lbu(0 BL5,3,0,347,or,,1,,,,
,,><&tbu2(1p3 BL1,9,327, 25p?,,,,,(n),(,100),,,,,,
,,>All text here<&te><&pbu2(-1p3 TL1,-9,1,(25p,R,7,0),,
,n,,(n),(,100),,K,,,m,,,,, " " ,><&g(8,7,6,5,4,3,2,1)>
```

This looks worse than it is. The translation table coding for this would be:

```
[[b1]]@RB:Review Box 12.1[[b2]]@rbtx:all text here[[b3]]
```

Expand or Shrink-to-Fit with Graduated Screen

This is a good use of use of the #16 field's "f" (expand or fit-to-size option). This is used in the picture box that brings in the graduated screen. It leaves the width at 100%, but reduces the depth of the screen to the depth of the text while maintaining the blend.



To automate this box:

1. Position the text box at 1p2 x and 2p4 y. The box will shrink to fit and you'll need a *Multiple Inset* of "(12,9,4,9)" for the text positioning.
2. Relative to this box will be the graduated screen that is an art file that needs to maintain the 100% width, but shrink-to-fit the depth to the bottom of the text. This is achieved by having the image import to the top left at -14 x and -24 y, selecting the "f" in the #16 field, having the "Send to Back" flag, and using relative placement and adding 1p7 to the depth.

```
<&pbu2(-14 TL1,-24,25p,(24p,R,1p7,0),,,k,n,,(n),(,100),,n,,f,,,,,
,, "56 GB Disk:Xtags Book:Images:Chapter 05:blend.eps" ,,,)>
```

3. The "Important Points" title will just position at 5p x and 0 y. We will also be adding the "gap" in the rule shading at 15% for the 4pt. double rule, and using the "centered" vertical alignment.

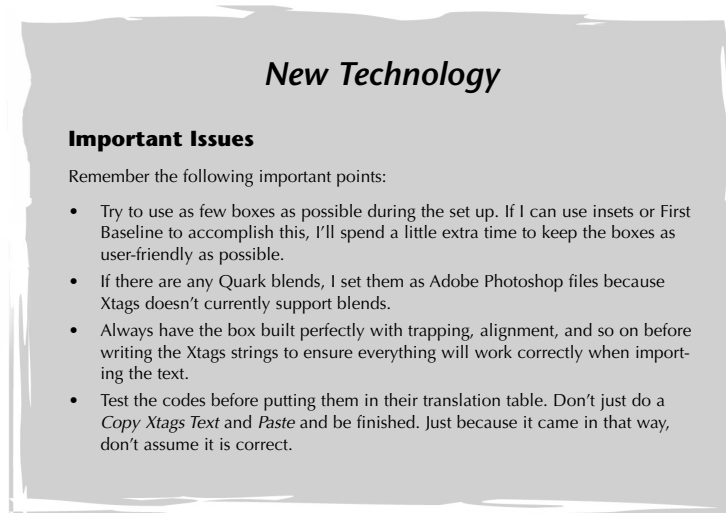
```
<&tbu2(5p B,0,15p9,33,,,,n,4,(K),(75,15),"Double" ,,,,,,c,,)>
```

4. The final code to make this box is:

```
<&tbu2(14 B,28,22p8,24p?,,,,n,,(n),(,100),,,,,,(12,9,4,9),,,,,)>Text goes
here<&te><&pbu2(-14 TL1,-24,25p,(24p,R,19,0),,,k,n,,(n),(,100),
,n,,f,,,,,"56 GB Disk:Xtags Book:Images:Chapter 05:blend.eps" ,,,)>
<&tbu2(55 B,0,189,33,,,,n,4,(K),(75,15),"Double" ,,,,,, c,,
,)>@bxt:Important Points<&te><&g(3,2,1)>
```

Box with 3 Separate Parts

Depending on the design, it is possible to find a box that will require being split into 3 separate parts to achieve automating. This is the case with the box element shown here. The rigid outline makes it too difficult to just have a top and bottom piece as seen in the “Side Margin Box Expandable” example.



To accomplish this box, you have to do a little playing around with the element to find the right place to break the image apart. Here are the steps:

1. The top of the image will remain as one section as seen here and will import first positioning on the 0 x, 0 y coordinate. No special handling.



2. The text will import next and will be 3p x, 3p y. This also has no special handling.
3. The bottom will also be its own section, which will not change size.



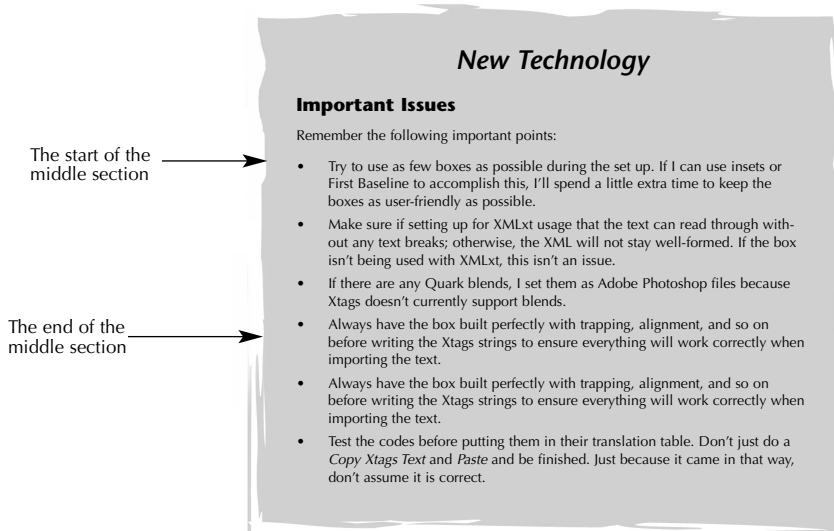
This will import -3p BL1 and -7p6 y. It will position using this tag:

```
<&pbu2(-3p BL1,-7p6.8,29p6,9p6.8,,,k,n,,(n),(,100),,n,,,m,,,,,
"56 GB Disk:Chapter 05:box_bottom.eps",,,)>
```

4. Where things get tricky is the middle section. This section needs to start at the base of the first piece of art, but extend only to the top of the bottom piece of art. This is done with relative placement. However, because of the rigid outline of the box shape, the top left corner needs to position exact, just as the bottom left corner. In order to do this, the tag needs to have relative placement with a -12p9.6 size reduction, and an “F” expand or shrink-to-fit #16 field option.

```
<&pbu2(-3p TL2,5p3,29p6,(30p,R,-12p9.6,0),,,k,n,,(n),(,100),
,n,,,F,,,,,,,"56 GB Disk:Chapter 05:box_middle.eps",,,)>
```

This is necessary because you want the middle piece of art to maintain 100% width, but the depth will be anywhere from 10% to 150% in size, but it will not be noticeable because the top and the bottom edges of the middle section line up perfectly to give the illusion that it is all one piece. Here's an example of how this will look with more bullet points. You can see the middle piece is still set correctly, but it is scaled at 62.5% depth by using the “F” #16 field option.



5. The full code for making this box is:

```
<&pbu2(0 B,0,29p6,99,,,,n,,(n),(,100),,n,,,m,,,,,,,"56 GB Disk:Chapter
05:box_top.eps",,,)><&tbu2(3p B,3p,24p,35p?,,,,n,,(n),(,100),,n,,,
,,,,,)>All box text goes here<&te><&pbu2(-3p BL1,-7p6.8,29p6,
9p6.8,,k,n,,(n),(,100),,n,,,m,,,,,,,"56 GB Disk:Chapter 05:
box_bottom.eps",,,)><&pbu2(-3p TL2,5p3,29p6,(30p,R,-12p9.6,0),
,,k,n,,(n),(,100),,n,,,F,,,,,,,"56 GB Disk:Chapter 05:box_middle.eps",
,,)><&g(1,2,3,4)
```

Unique Box Treatment Using Skew and Angle

With some inventiveness, a person can really do some surprising things with Xtags and try to go beyond the normal limitations. Here's an example of a box that most pagers would look at and most likely make a two part piece of art.

"Kids play too many video games and watch too much television. They don't know what it's like to just be bored anymore."

—William R. McKinley

Some would think this would have to be a two part Xtags picture box using an Illustrator file where the top half would accommodate the longest quote, and the bottom will import with relative placement based on the length and close it off similar to the "Side Margin Box Expandable" example. That will work, but surprisingly, this can be built in Quark without the assistance of art programs. To do this.

1. The text box will be a standard shaded text box with rounded corners and a *First Baseline* of 2p. Additional depth will need to be added to the shrink-to-fit on field #4.
2. The bottom triangular shaped box is a combination of two fields: the *boxangle* #5 field, and the *boxskew* #6 field, but also with the "Send to Back" flag. Combining the two creates the following shape:



The code for this rotated and skewed box is:

```
<&pbu2(47 BL1,-28,8p,2p6,31,24,k,n,,(n),(,100),,K,15,,m,,,,,"" ,,,)>
```

3. Upon the shrink-to-fit, this will need to position exact to give this impression. The final code is:

```
<&tbu2(0,0,268,24p?p7,,,,n,(,36),(,n),(,100),,K,15,,,,2p2,,,,)>All  
text here<&te><&pbu2(47 BL1,-28,102,37.9,31,24,k,n,,(n),(,100),  
,K,15,,m,,,,,"" ,,,)><&g(2,1)>
```

Using Xtags for Design Element in Box

Do not be afraid to use Xtags to add flair to a box. There are times when a pager sees a box set up with a design element and end up creating the element in an art program. This is workable, but there are times when this can be built entirely in Quark. When using a translation table, the design element can all be part of one code. Consider this box as an example:

Steps to Follow in Box Building

Follow these steps when setting up boxes:

- Try to use as few boxes as possible during the set up. If I can use insets or First Baseline to accomplish this, I'll spend a little extra time to keep the boxes as user-friendly as possible.
- Always have the box built perfectly with trapping, alignment, and so on before writing the Xtags strings to ensure everything will work correctly when importing the text.

When first looking at the upper left design element, this appears that it would be less work to create this in Adobe Illustrator and then import it in the corner each time. This is a less involved way to do this, but you can develop your skills by using Xtags to create this. To do this with Xtags instead of as art:

1. The element is comprised of five different picture boxes. The line tags could be used for this as well. This needs to be broken down by starting with the outside 15% screened horizontal and vertical boxes. These are 5p in length and 5 points in width. The inside lines are 3p6 in length and 5 points width. The small square box in the upper left corner is 5 points length and width.
2. The text box needs to have the "Send to Back" flag and begins at an x and y coordinate where the outer rule splits the outside design boxes. This is done by using a 1.5 x and 1.5 y offset.
3. The x and y coordinates need to be precise because you do not want any white space between the design element lines. Pay attention at how each of the first five boxes position on the x and y of either 0, 5, or 10 points. Here is an example of the strings put together:

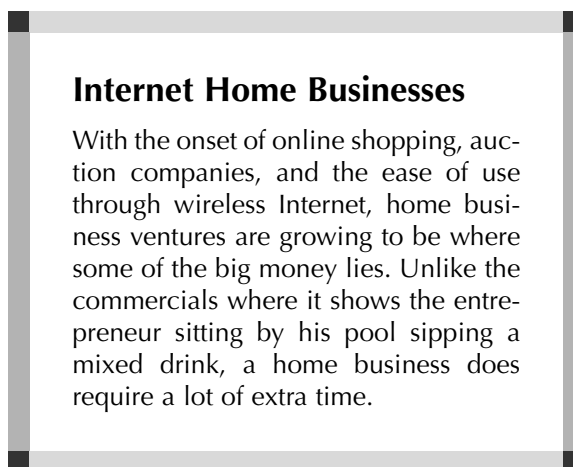
```
<&pbu2(0,0,60,5,,,n,,(n),(,100),,K,15,,m,,,,,"",,)><&pbu2(10,5,42,5
,,,n,,(n),(,100),,K,25,,m,,,,,"",,)><&pbu2(5,10,5,42,,,n,,(n),(,100),,
K,25,,m,,,,,"",,)><&pbu2(0,0,5,60,,,n,,(n),(,100),,K,15,,m,,,,,"",,)>
<&pbu2(5,5,5,5,,,n,,(n),(,100),,K,50,,m,,,,,"",,)><&tbu2(1.5,1.5,24p,
30p?,,,k,n,2,(n),(20,100),,n,,,,(14,12,8,9),,,,,)>All box text here<&te>
<&g(1,2,3,4,5,6)>
```

4. The markup using a translation table will be broken down to 2 tags:

```
[[t1]]The box text goes here[[t2]]
```

Using all the Placements in One Element

When first looking at the box below, it may not seem all that complicated, but I assure you that if you build this properly, you will get to use BL1, BR1, TL1, TR1, and relative placement on each side. This is a great test to see if you can figure it all out.



This box requires a lot of planning to be correct. Here are the steps:

1. The text area will start on the coordinates of 2p x and 2p y. This makes figuring out the positioning of the other boxes much easier.
2. The next box I will concentrate on will be the top left 8 point square. This will be -2p TL1 x and -2p y.
3. The 15% box to the right will be handled with relative placement to the text box. Since we know the width of the text, it would seem more obvious to just make this a standard size, but we are going to pretend that the text width is flexible where it can be marginal or in the text area. That makes this easier to keep this adjustable.

The text area has a maximum width of 32p, so the relative placement width will be (28p,R,2p8,0) with a 2p8 additional width to accommodate the -1p4 x and the additional 1p4 width required. Since the text is inset 2p on the left and right, 28p will be the maximum width the text box can be. This box reads off of the text box width, so this will be using -1p4 TL2 x with -2p y:

```
<&pbu2(-1p4 TL2,-2p,(28p,R,2p8,0),8,,,n,,(,n),(,100),,K,15,
,m,,,,,, " " ,,,)>
```

4. Our next box will be the upper right square. This will also read off of the text box requiring a 1p4 TR3 x and -2p y coordinates.

```
<&pbu2(1p4 TR3,-2p,8,8,,,n,,(,n),(,100),,K,80,,m,,,,,, " " ,,,)>
```

136 *Xtags Maximized*

5. The left vertical box running the length of the text is next, which sits at the coordinates of -2p TL4 x (off of the text again), and -1p4 y. This will also be relative to the length of the text, but will have 2p6.5 additional depth added because the descenders on the text extend the box by 1.5 points.

```
<&pbu2(-2p TL4,-1p4,8,(30p,R,2p6.5,0),,,,n,,(n),(,100),,K,30,
,m,,,,,"")>
```

6. The bottom bar and the right bar are very similar to their box parallel from it. The bottom right square will require a 1p4 BR7 offset from the text box.

```
<&pbu2(1p4 BR7,1p2.5,8,8,,,n,,(n),(,100),,K,80,,m,,,,,"")>
```

7. The complete tags for this will be:

```
<&tbu2(2p B,2p,14p,30p?,,,n,,(n),(,100),,n,,,,,,>Text
Here<&te><&pbu2(-2p TL1,-2p,8,8,,,n,,(n),(,100),,K,80,,m,,,
,"")><&pbu2(-1p4 TL2,-2p,(32p,R,2p8,0),8,,,n,,(n),(,100),,
K,15,,m,,,,,"")><&pbu2(1p4 TR3,-2p,8,8,,,n,,(n),(,100),,K,80,,m,
,,,,,"")><&pbu2(-2p TL4,-1p4,8,(30p,R,2p6.5,0),,,,n,,(n),(,100),,
K,30,,m,,,,,"")><&pbu2(-2p BL5,1p2.5,8,8,,,n,,(n),(,100),,K,80,
,m,,,,,"")><&pbu2(-1p4 BL6,1p2.5,(32p,R,2p8,0),8,,,n,,(n),(,100),,
K,15,,m,,,,,"")><&pbu2(1p4 BR7,1p2.5,8,8,,,n,,(n),(,100),,K,80,
,m,,,,,"")><&pbu2(1p4 TR8,-1p4,8,(30p,R,2p6.5,0),,,,n,,(n),
(,100),,K,30,,m,,,,,"")><&g(1,2,3,4,5,6,7,8,9)>
```

8. The coded file for this will be very simple using only two tags:

```
[[b1]]Box text[[b2]]
```

9. The relative placement for each box was necessary so if we change the text width from 14p to 23p, the other boxes will not have to be adjusted throughout. This is the box importing with a 23p text width:

Internet Home Businesses

With the onset of online shopping, auction companies, and the ease of use through wireless Internet, home business ventures are growing to be where some of the big money lies. Unlike the commercials where it shows the entrepreneur sitting by his pool sipping a mixed drink, a home business does require a lot of extra time.

A graphic for Chapter 6 titled 'Custom Publishing'. It features a large, stylized number '6' in a metallic, 3D font on the left. To its right, the words 'Custom Publishing' are written in a large, bold, white sans-serif font. Below the '6', the word 'CHAPTER' is written in a smaller, white, all-caps sans-serif font. The background is dark gray with a grid of thin, light gray lines and some subtle light effects.

6 Custom Publishing

CHAPTER

In the changing college textbook environment, custom publishing has become a big market. Authors are now letting professors pick and choose which chapters they want to use in their curriculum. Sometimes this isn't just one book in particular. It could be several books by different authors. This gives the compositor a major challenge. How do you take chapters from 4 different titles and make them look like the one book? How do you take chapters from Quark 6.5 or 7 and bring that content into Quark 4.1.1? All the while doing this within the required budget. My answer is by utilizing Xtags and BBEdit.

I have done many books like this and I have found that I can successfully execute this by tapping into the power of Xtags. When doing a custom publishing job, it's imperative to study the material before starting the project. I begin by asking myself the following questions:

1. What is the book that all chapters will need to look like?
2. Are there any size/margin differences that will affect the art?
3. Are the fonts the same or will these need to change?
4. Will the point size change?
5. Did they use the same style sheets for naming similar content?
6. Are there new elements that will have to be designed for?
7. Is the end matter elements consistent, or will they need altering?
8. Are the references such as Figure 1.1, Table 1.1, etc consistent or do the other books use hyphens between the chapter and figure number.
9. Are there references to other chapters or figures?
10. If the book is 4-color, which colors will need to change for consistency to which new colors?

These are the main concerns I start with. Others may appear, but once these are addressed, you are pretty much on your way to working on the title and having some consistency. Missing or inconsistent content is one of the hardest obstacles, but we'll cover that here soon. We are going to be covering

Grep searches in BBEdit, so it's best to take a look at this before we get too deep into this.

Some Background on Grep Searches

One of the greatest features in the “Find & Replace” window is using the *Grep* searches. This is very similar to writing a MacPerl string with the *s///g*; pattern replacement. There are some differences in code structure that are easy to become familiar with. *Grep* is very involved and teaching every facet of it would take up more pages than I have available in this section. My goal here is to provide an overview of its capabilities.

The idea is to match patterns and be able to ignore content that would be included in a normal “search and find.” I want to point out that the majority of my changes are implemented using “subpatterns.” “Subpatterns” are groups of *Grep* separated by parentheses () that can be manipulated and used to reorder, delete, or add content between the groupings. These are handled differently than in MacPerl. In MacPerl, if I have a grouping like:

```
s/(\\@FIG:)(.+?)(\\n)/$1$2\\[f1]]$3/g;
```

The subpatterns are \$1, \$2, \$3, and so on. For *Grep* these are tagged as \\1, \\2, \\3, and so on. If you have some understanding of MacPerl, *Grep* should come fairly easy to you. Learning to use *Grep* is power. There is very little you cannot change throughout a document or multi-file replacement.

I will show you a couple of examples of some amazing code changes that can be achieved just by understanding many of the features. Again, it is extremely important to delimit (\\) the following characters unless you are using these as functions in the “Find & Replace” window:

```
+ ? \ / | * ^ . [ ( ) $ &
```

This list is not complete, but makes one aware of what will need a “\\” preceding it. These all have functions within the *Grep* patterns. Here are a few examples:

- ^ Match at the beginning of a line
- \\ Used with subpatterns such as \\1, \\2 and so on.
- () Used for grouping subpatterns
- . Any character
- \$ Match at the end of a line
- | Separates two functions within a subpattern
- & Will replace the entire match pattern

If unsure whether a character should be delimited, it's always a good idea to delimit it or test to see if the results happen by not delimiting. In most cases, there is more than one way to achieve the results. For example, if needing

to find every occurrence of `[[AR ID#]]`, you could try to find it a couple different ways:

<code>(\\[AR](\\d\\d\\d)(\\))</code>	This will find either a single or double digit in the second subpattern
<code>\\[AR \\d+]</code>	This will find any number until reaching the closing <code>]</code>
<code>\\[AR.+?]</code>	This finds any character between the “AR” and the closing “ <code>]</code> ”

Any of these would work in this situation; however, I find that the second example is generally the best for my needs.

Looking at the Operators

Grep searches provide so many different ways of matching on content through the wildcards and characters. It is important to learn what these operators are to make the experience one that can speed up the project's production time.

Let's look at a few ways to match on something. Let's say we need to find the callouts for all tables in Chapter 10 such as “Table 10.1” and change them to be tables for Chapter 9 such as “Table 9.1”. This is accomplished by looking for finding the “9”. We do not want to match on every “9” in the chapter. There could be different scenarios where we have text like “9 miles to town” or “394” or “She is 19 years old.” The match needs to be more specific.

To match on a number, there are several different options:

<code>\\d</code>	Any single digit number
<code>\\d+</code>	Any number infinitely
<code>[0-9]</code>	Any single digit number
<code>[0-9]+</code>	Any number infinitely
<code>[:digit:]</code>	Any single digit number
<code>[:digit:]+</code>	Any number infinitely
<code>\\w</code>	Any word character (0-9, a-z, A-Z, etc.)
<code>\\S</code>	Any non-whitespace character (except space, tab, etc.)

This leaves us with a number of options. I typically will go with the easiest solution which would be the “`\\d+`” in this case. To find “Table 10.1”:

Search for:

`(Table)(\\d+)(\\.\\d+)`

Replace with

Table 9\\3

I wrote out the word “Table” because I couldn't have `\\19\\3` because *Grep* would interpret this as `\\19` rather than `\\1` and a “9”.

140 *Xtags Maximized*

Let's look at another possibility. If we needed to look through the document to find the font for "<f\"Berkeley-Italic">, <f\"Berkeley-Medium">, and <f\"Berkeley-Bold">" and delete these references, we could have several different ways to handle this.

\w	Any word character (0-9, a-z, A-Z, etc.)
[a-z]	Any lowercase letter
[A-Z]	Any uppercase letter
[A-Za-z]	Any lower or uppercase letter
[A-Za-z]+	Any lower or uppercase letter until proven false
[:alpha:]	Any letter
\S	Any non-whitespace character (except space, tab, etc.)

In this case we are left with the option of which way to go here. We can do a search for all 3 with a pipe "|" between them or handle it the most obvious way using the "\w" option. I feel, in this case, it is the most direct approach.

Search for:

<f\"Berkeley-\w+\\">)

Replace with

Empty

This works well because it will find all the different scenarios of "Berkeley" that may not have been considered. If you wanted to keep some options of "Berkeley" such as "Bold Italic", then you would want to use the pipe and be very specific.

Another great example is if you have marginal notes and the first character was all set with lowercase by the typist, but after reviewing the design, the editor realized they should all be uppercase. Instead of going through and changing each instance by hand, you could simply do the following

Search for:

(\@mn1:)([a-z])(.+?)\r)

Replace with

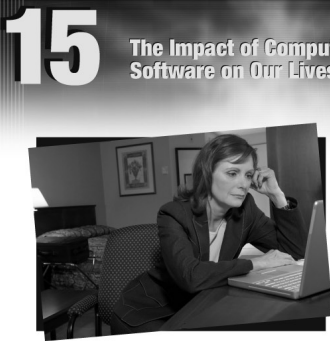
\1\u\2\3\4

The "\u" is replacing the first lowercase letter following the style name to be an uppercase letter.

This should give you some basic background on *Grep* searches. I am barely skimming the surface in here on what you can achieve with *Grep*. Performing *Grep* searches are very similar to the Perl type of coding if you do that. For more detailed information, look through the *BBEdit User Manual* which has an extensive section concerning *Grep*.

Custom Publishing Using Xtags

So what is the first step to doing custom publishing using Xtags? To demonstrate this, here is an example of two chapters. The Book 1 (Top) is the style for the book. Book 2 (Bottom) will need to change to match the Book 1.



Chapter Objectives

After studying this chapter you will be able to:

1. Understand the relationship between computer software and how it impacts our daily lives from commuting to leisure time.
2. The differences between software and hardware
3. How the Internet and other forms of communication such as Instant Messaging has changed the way we communicate in business
4. Describe some methods companies can use to strengthen their business using the Internet and email
5. Appreciate the important ways in which we look for new ways to take software to quicken tasks by using scripts and other automation methods

The Impact of Computer Software on Our Lives 139

Overview of Computer Software

Xtags: A valuable QuarkPress Xtension that works with the code underneath that you cannot see. It works for text and images and can speed up your workflow in the way that reads. Xtensions only dream of.

One of the prerequisites to be successful with this book is that you have used Xtags before, preferably on an actual project. Also, I feel it's necessary that you have read through the Xtags Manual by En Software, Inc., which is an invaluable reference guide. It's obvious to me that there would not be a need for this book if I didn't have a distinguishing approach to the material. Where I feel this book is unique is through my step-by-step approach, helping you achieve the task without having to keep referring back to certain sections to pull it off. I always have learned through intuition like this and I always retain something once I do it. It locks in the memory better than just reading about descriptions of various tasks.


I also want to help people who sometimes get a little confused on where to start. It's one thing to set up a book in Autopage, but it's another to get the Xtags, Headers & Contents, XML, and even the QuarkPress document set up properly. The more you have working correctly up front, the less chance there will be of unforeseen problems later in the project.

This book has not covered every aspect of Autopage because that is not my intention. My main objective is to show the user how to get more out of the program by approaching the program in a different manner than the user might be used to. After following through the book and learning how to do all of the procedures, Chapter 15 is full of step-by-step examples that will give you an opportunity to apply the various functions of the program. I feel it intermediates and advanced users work through these, they'll find themselves more proficient than before. Many of these concepts you'll already know, but my hope is that you'll learn some new and important things along the way.

The New Direction of Software

Since I've been using the program, I've noticed there are many misconceptions about Autopage and its uses. It can be used in many areas of publishing, but does not necessarily take away all of the hardwork. If anything, it reduces time spent on manual tasks, but there will still be a need for the pages to get involved and teach things up. This is probably a good thing because otherwise there would not be a need for paggers, so this is job security.

Some of the biggest arguments that paggers always use to avoid using the program are shown in this section. I must stress that most of these statements have come directly from paggers who haven't spent enough time learning the program and continue doing manual tasks that should be automated.



Setting up the Project


This could be seen as a justifiable excuse to some degree, especially when it is the heaviest time of the production season and you're in the middle of several books. It is sometimes difficult to put aside the necessary time to focus and complete the up-front tasks. However, if the book isn't set up with Autopage, a massive amount of wasted time will be spent manually creating layout changes, putting in margin notes, and positioning art. The up-front time spent on coding the sample would have been saved throughout the project. In my opinion, a successful, cost-effective project relies heavily on the up-front setup of Xtags.

How far you go with the Xtension is up to you. You can begin using the software the first day and gain more knowledge with each book or you can explore functions whenever you have time and amaze others at your findings. Once you get going forward, you'll find that tasks you normally would have completed manually are now handled this way.

Book 1 (The Style for the Entire Book)

Can Second Party Software be the Best Investment?

chapter 9



Objectives

- Understand the relationship between computer software and how it impacts our daily lives from commuting to leisure time.
- The differences between software and hardware
- How the Internet and other forms of communication such as Instant Messaging has changed the way we communicate internally within businesses
- Describe some methods companies can use to strengthen their business using the Internet and email
- Appreciate the important ways in which we look for new ways to take software to quicken tasks by using scripts and other automation methods

Can Second Party Software be the Best Investment? 141

Intro to Second Party Xtensions

Second party Xtension have revolutionized QuarkPress and may be a big sell in the future for InDesign. I always had a good grip on QuarkPress, but when I needed that boost of power, I knew there was a developer out there who had just the thing I needed. I would go out of my way to learn this software. I also want to help people who sometimes get a little confused on where to start. It's one thing to set up a book in Autopage, but it's another to get the Xtags, Headers & Contents, XML, and even the QuarkPress document set up properly. The more you have working correctly up front, the less chance there will be of unforeseen problems later in the project.

What is Second Party Software?

Xtags: A valuable QuarkPress Xtension that works with the code underneath that you cannot visibly see. It works for both text and images.

One of the prerequisites to be successful with this book is that you have used Xtags before, preferably on an actual project. Also, I feel it's necessary that you have read through the Xtags Manual by En Software, Inc., which is an invaluable reference guide. It's obvious to me that there would not be a need for this book if I didn't have a distinguishing approach to the material. Where I feel this book is unique is through my step-by-step approach, helping you achieve the task without having to keep referring back to certain sections to pull it off. I always have learned through intuition like this and I always retain something once I do it. It locks in the memory better than just reading about descriptions of various tasks. See Figure 9-1.

This could be seen as a justifiable excuse to some degree, especially when it is the heaviest time of the production season and you're in the middle of several books. It is sometimes difficult to put aside the necessary time to focus and complete the up-front tasks. The up-front time spent on coding the sample would have been saved throughout the project. In my opinion, a successful, cost-effective project relies heavily on the up-front setup of Xtags.

This book has not covered every aspect of Autopage because that is not my intention. My main objective is to show the user how to get more out of the program by approaching the program in a different manner than the user might be used to. After following through the book and learning how to do all of the procedures, Chapter 9 is full of step-by-step examples that will give you an opportunity to apply the various functions of the program. I feel it intermediates and advanced users work through these, they'll find themselves more proficient than before. Many of these concepts you'll already know, but my hope is that you'll learn some new and important things along the way.

The Impact on Profitability

Since I've been using the program, I've noticed there are many misconceptions about Autopage and its uses. It can be used in many areas of publishing, but does it

Book 2

As you can clearly see, the chapters are very different. On the opening page, we see the following differences immediately:

1. The first book contains an objectives with numbers and the second book has objectives using colored square bullets.
2. You'll notice that the Objectives title is different on both.
3. There is a lead-in sentence on the first book, where the second fails to put this in.
4. The first book has an opening image that is centered and slightly rotated on the page with a block shadow behind it, whereas the second book has an image taking up most of the margin area.
5. The first book has the chapter identified as only "15" whereas the second is "Chapter 9".

I tried to make this as simple as possible by keeping the content close to the same. This isn't usually the norm as when doing custom publishing you'll find new content all over the place that will need to be designed for or altered slightly.

The Opening Page

For starters, I will begin by addressing the opening page and using BBEdit combined with AppleScript to make these adjustments. Some people may think it would be easier to just copy and paste in the text and then find the right style sheet. This thinking opens up the doors for far too many issues. When pasting in, it's too easy to change the font and not change the point size, or leave tracking on that should be removed. There are too many variables that need to be addressed. Don't leave yourself wide open to errors.

Typically, you will have more than one chapter that you are going to be using from other books. For that reason, whenever I'm doing *Grep* searches in BBEdit, I find myself making an AppleScript to record the replacements so they can be reused throughout the process. Let's start by going through this process just starting with the first page. Typically I do the entire document, but for learning purposes, we need to start slow:

1. First, go through and make sure the images are all updated.
2. Go through and select the text from the chapter number to the end of the objectives. Then select *Edit:Copy Xtags Text*.
3. Open up BBEdit, and paste this information into a new document. For this title, the text appeared as follows:

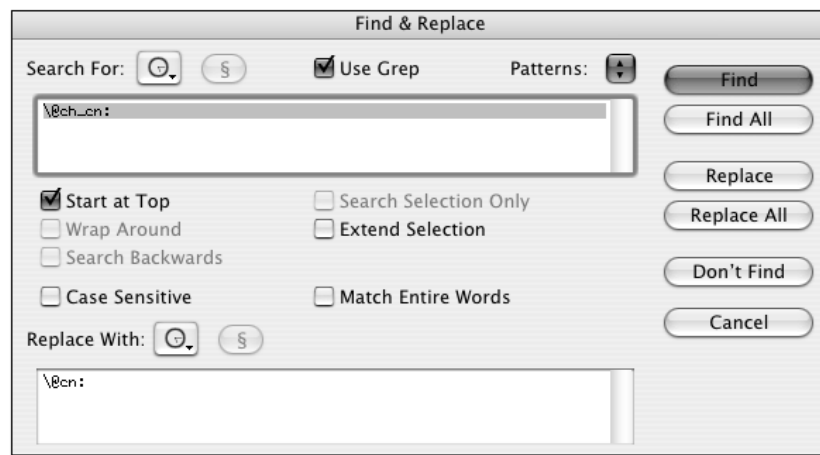
```
@ch_cn:9
@ch_ct:Can Second Party Software be the Best Investment
@ch_objt:Objectives
```

```

@ch_obj:<z9.5c"Red01"f"ZapfDingbats">n<z11cKf"NewBaskerville-
Roman">Understand the relationship between <f"Times-
Italic">computer software<f"Times-Roman"> and how it impacts our
daily lives from commuting to leisure time
<z9.5c"Red01"f"ZapfDingbats">n<z11cKf"NewBaskerville-Roman">
The differences between software and hardware
<z9.5c"Red01"f"ZapfDingbats">n<z11cKf"NewBaskerville-Roman">
How the Internet and other forms of communication such as Instant
Messaging has changed the way we communicate internally within
businesses
<z9.5c"Red01"f"ZapfDingbats">n<z11cKf"NewBaskerville-Roman">
Describe some methods companies can use to strengthen their
business using the Internet and email
<z9.5c"Red01"f"ZapfDingbats">n<z11cKf"NewBaskerville-Roman">
Appreciate the important ways in which we look for new ways to
take software to quicken tasks by using scripts and other automation
methods
<&pbu2(57,167,480,347.592,,,,n,,(K,n),(100,100),"Solid",W,100,0,m,,,,,
,"56 GB Disk:Xtags Book:Images:ch09_image1.eps",,,,"")>

```

4. The first thing that is most noticeable here is that the style names are going to be different than the original book and that the objectives have the bullets which will need to be removed. To do this, it is best to do *Grep* searches or write these in AppleScript. We'll start by doing a *Grep* search through BBEdit that will change these.



While *Grep* is extremely useful, to do this to multiple chapters, the most effective method that I suggest is to do this in AppleScript. This is the smartest way to go because it will save time on future chapters by making all of these changes at once.

144 *Xtags Maximized*

Before we get into all the changes, it's important for you to understand a little bit on how the *Grep* searches work with AppleScript.

Shown below are all of the changes that will need to happen in this section of text. The final *AppleScript* for this is as follows:

```

tell application "BBEdit"
    activate
    replace "\\@ch_cn:" using "\\@cn:" searching in text 1 of text
        document 1 options {search mode:grep, starting at top:true,
        wrap around:false, backwards:false, case sensitive:false,
        match words:false, extend selection:false}
    replace "\\@ch_ct:" using "\\@ct:" searching in text 1 of text
        document 1 options {search mode:grep, starting at top:true,
        wrap around:false, backwards:false, case sensitive:false,
        match words:false, extend selection:false}
    replace "\\@ch_obj:" using "\\@obj:" searching in text 1 of text
        document 1 options {search mode:grep, starting at top:true,
        wrap around:false, backwards:false, case sensitive:false,
        match words:false, extend selection:false}
    replace "(<f\\\\"NewBaskerville-
        Italic\\\\">)(.+?)(<f\\\\"NewBaskerville-Roman\\\\">)" using
        "<I>\\2<\\$>" searching in text 1 of text document 1 options
        {search mode:grep, starting at top:true, wrap around:false,
        backwards:false, case sensitive:false, match words:false,
        extend selection:false}
    find "<z9\\.5c\\\\"Red01\\\\"f\\\\"ZapfDingbats\\\\">n
        <z11ckf\\\\"NewBaskerville-Roman\\\\"> " searching
        in text 1 of text document 1 options {search mode:grep,
        starting at top:true, wrap around:false, backwards:false, case
        sensitive:false, match words:false, extend selection:false} with
        selecting match
    grep substitution of "1.\\t"
    set selection of window 1 to "1. "
    try
        find "<z9\\.5c\\\\"Red01\\\\"f\\\\"ZapfDingbats\\\\">n
            <z11ckf\\\\"NewBaskerville-Roman\\\\">\\t" searching in
            text 1 of text document 1 options {search mode:grep, starting
            at top:false, wrap around:false, backwards:false, case
            sensitive:false, match words:false, extend selection:false} with
            selecting match
        grep substitution of "2.\\t"
        set selection of window 1 to "2. "
    end try
end tell

```



```

try
    find "<z9\\.5c\\.\\.Red01\\.\\.f\\.\\.ZapfDingbats\\.\\.>n
    <z11cKf\\.\\.NewBaskerville-Roman\\.\\.>\\.t" searching in
    text 1 of text document 1 options {search mode:grep, starting
    at top:false, wrap around:false, backwards:false, case
    sensitive:false, match words:false, extend selection:false} with
    selecting match
    grep substitution of "3.\\.t"
    set selection of window 1 to "3. "
end try
try
    find "<z9\\.5c\\.\\.Red01\\.\\.f\\.\\.ZapfDingbats\\.\\.>n
    <z11cKf\\.\\.NewBaskerville-Roman\\.\\.>\\.t" searching in
    text 1 of text document 1 options {search mode:grep, starting
    at top:false, wrap around:false, backwards:false, case
    sensitive:false, match words:false, extend selection:false} with
    selecting match
    grep substitution of "4.\\.t"
    set selection of window 1 to "4. "
end try
try
    find "<z9\\.5c\\.\\.Red01\\.\\.f\\.\\.ZapfDingbats\\.\\.>n
    <z11cKf\\.\\.NewBaskerville-Roman\\.\\.>\\.t" searching in
    text 1 of text document 1 options {search mode:grep, starting
    at top:false, wrap around:false, backwards:false, case
    sensitive:false, match words:false, extend selection:false} with
    selecting match
    grep substitution of "5.\\.t"
    set selection of window 1 to "5. "
end try
end tell

```

You'll notice the sections broken by "try" and "end try" are actually converting the bullets into numbers. This can be handled in many different ways, but this is the easiest to understand at this point. What it is doing is for the first one, it goes to the top of the BBEdit text file by keeping the "starting at top:true" selected. It then matches the first bullet and changes it to be "1".

Following this I change the selection to be "starting at top:false" which goes to the one immediately after and changes it to "2.", then "3." and so on. The only problem you could have here is if you do an entire chapter and this exact match would be found later. You would then run into issues. I know this

particular book only has up to five bullets per chapter, so this keeps it from going beyond the fifth bullet if one exist elsewhere.

After running the AppleScript against this text, the text file will now be converted to the following:

```
@cn:9
@ct:Can Second Party Software be the Best Investment
@objt:Chapter Objectives
@obj_tx:After studying this chapter you will be able to:
@obj:1. Understand the relationship between <l>computer
      software<$> and how it impacts our daily lives from commuting
      to leisure time
2. The differences between software and hardware
3. How the Internet and other forms of communication such as
      Instant Messaging has changed the way we communicate
      internally within businesses
4. Describe some methods companies can use to strengthen their
      business using the Internet and email
5. Appreciate the important ways in which we look for new ways to
      take software to quicken tasks by using scripts and other
      automation methods
```

I also made a change on the opening image by importing it already rotated with the “fit-to-width” additional parameter in the #3 field. The sizing will be completed for me with the proper size reduction. Here are the changes:

```
<&pbu2(0,0,(31p9,f),26p?,5,,,n,,(K,n),(100,100),"Solid",W,100,0,m,,,,
,,,56 GB Disk:Xtags Book:Images:ch09_image1.eps",,,,")>
```

The next section we are going to look at is a page of the text area. This is very much with the same type of treatment. There’s also a new element that was not part of the first book. It’s a vignette at the top of the second page that says “Intro to Second Party Xtensions”. It appears like this:

Intro to Second Party Xtensions

Second party Xtension have revolutionized QuarkXPress and may be a big sell in the future for InDesign. I always had a good grip on QuarkXPress, but when I needed that boost of power, I knew there was a developer out there who had just the thing I needed. I would go out of my way to learn this software. I also want to help people who sometimes get a little confused on where to start. It’s one thing to set up a book in Autopage, but it’s another to get the Xtags, Headers & Contents, XMLxt, and even the QuarkXPress document set up properly. The more you have working correctly up front, the less chance there will be of unforeseen problems later in the project.

This will need to be designed for. This is where having a Design Style Database in FileMaker Pro will be of benefit. That way you don't have to really spend very much time thinking about it. This is covered in the last chapter.

So the first thing I will do here is to design all of the missing elements for this first chapter. I know this may seem time consuming, but it's more cost-effective to deal with this now, rather than doing it when you are in the paging process and you lose your momentum. So, let's first create this. Even if you are not a fantastic designer, remember to keep the elements consistent and with the same feeling of the book. So with this, I will get the look down. I will actually create the finished result using Xtags each time one of these show up in a chapter to keep it together. Here is the Xtags code:

```
<&tbu2(36 B,0,31p,24p8?,,,,n,0,(K,n),(100,100),"Solid",n,
100,,1,,,,t,,,"")>@vig1:Intro to Second Party Xtensions
@vig:Second party ... project.<&te><&pbu2(-22 TL1,0,9,(177,R,0,0),
,,,,(K,n),(100,100),"Solid",n,100,0,f,,,,,"56 GB
Disk:Xtags:Images:vignette2.eps",,,,"")><&g(2,1)>
```

There is now a decorative bar down the side that shrinks to fit with relative placement and font has changed. Here is how an example of how this will now appear in the first book:

Intro to Second Party Xtensions

Second party Xtension have revolutionized QuarkXPress and may be a big sell in the future for InDesign. I always had a good grip on QuarkXPress, but when I needed that boost of power, I knew there was a developer out there who had just the thing I needed. I would go out of my way to learn this software. I also want to help people who sometimes get a little confused on where to start. It's one thing to set up a book in Autopage, but it's another to get the Xtags, Headers & Contents, XMLxt, and even the QuarkXPress document set up properly. The more you have working correctly up front, the less chance there will be of unforeseen problems later in the project.

I would put this information in a translation table where the actual text file would appear like this.

```
[[vig]]@vig1:Intro to Second Party Xtensions
@vig:Second party ... project. [[vig2]]
```

For the remainder of this page, do the following:

1. Copy the text and select *Edit:Copy Xtags Text*.
2. Paste into a new BBEdit file.

The text exported from the Quark file is:

```
@ch_h1:What is Second Party Software?
@ch_tx1:One of the prerequisites ... than just reading about
descriptions of various tasks. See Figure 9–1.
@ch_tx:<t-1>This<k-2> <k0>could<k-2> <k0>be<k-2> <k0>seen<k-
2> <k0>as<k-2> <k0>a<k-2> ... <k0>setup<k-2> <k0>of<k-2>
<k0>Xtags.
@ch_mn:<@ch_mnkt>Xtags<\f><@$p>A valuable QuarkXPress
Xtension that works with the code underneath that you cannot
visibly see. It works for both text and images.
@ch_fgcp:<BK>Figure 9–1<BK><\f><z7b0.5f"ZapfDingbats">n
<z9b0f"Helvetica"><\f><\i>Leisure time for couples is spent more
frequently using laptops.
```

3. The paragraphs above show a lot of overrides and manual manipulation by the pager opposed to creating character style sheets. Some of this could not be helped. For example, in the “@ch_tx:” paragraph that shows all the lines with the kerning and tracking:

```
<t-1>This<k-2> <k0>could<k-2>
```

This was not done manually by the pager, but rather by the Autopage program. This, of course, you would want to remove since the style and fonts are changing. This is fairly easy to do without losing anything. The method to do this is by doing a *Grep* search and including the following in the “Search For”:

```
<t-\d+>|<\d+>|<k\d+>|<k-\d+>
```

You would need to leave the “replace” as blank.

This this would also work in an AppleScript by including this line:

```
tell application "BBEdit"
    activate
    replace "<t-\d+>|<t\d+>|<k-\d+>|<k\d+>" using "" searching
        in text 1 of text document 1 options {search mode:grep,
            starting at top:true, wrap around:false, backwards:false,
            case sensitive:false, match words:false, extend
            selection:false}
end tell
```

This is actually a line you would want to include in all of your AppleScripts for custom publishing. If the tracking is necessary, it should be in the style sheets. You will want to always review your content ahead of time (especially

in annotated English books) to make sure the pager didn't do strange tracking that was needed as a manual override.

There is also a font issue that needs resolved. In the first line of the "ch_tx1" line, it shows:

```
<f"NewBaskerville-Italic">Xtags Manual<f"Times-Italic">
<f"NewBaskerville-Roman">
```

This happened because of a manual override where the "Times-Italic" font is on a space and shouldn't even be in the paragraph. The way I would try to delete this is by deleting it in that line and hoping it would catch more errors like that if they would occur.

Search for:

```
(<f"NewB.+?lle-Italic">)(.+)(<f"Times-Italic">)( |.+)
(<f"NewB.+?lle-Roman">)
```

Replaced with:

```
<l>\2\4<$>
```

This will give you this result:

```
<l>Xtags Manual<$>
```

The figure legend was originally formatted as:

```
@ch_fgcp:<BK>Figure 9-1<BK><\f><z7b0.5f"ZapfDingbats">n
<z9b0f"Helvetica"><\f><i>Leisure time for couples is spent more
frequently using laptops.
```

It needs to be:

```
@fgc:<@fgn>Figure 9.1<@$p><\f><\f>Leisure time for couples is
spent more frequently using laptops.
```

To achieve this, a *Grep* search needs to happen as:

Search for:

```
(\\@ch_fgcp:)(<BK>)(Figure \\d+)(-)(\\d+)(<BK>)(<\\f>)
(<z7b0.5f"ZapfDingbats\\">n<z9b0f"Helvetica\\">)(<\\f>)(<\\i>)
```

Replaced with:

```
\\@fgc:<\\@fgn>\\3\\.\\5<\\@$p>\\7\\9
```

The secret here is in the parenthesis and putting just enough in them to make them unique. You don't want to include too much. The more you put in the more difficult it will get to figure out exactly what content needs changing.

You don't want to leave out content either. If you don't make sure it's unique, it could go through the chapter and replace lines that you don't want to touch.

The style sheet changes were covered before, but one of the largest issues seen here is in the figure caption. The figure caption for this original book is much different than the one we are going to. You will also want to address the fact that the callouts are using en-dashes rather than periods. These will need to be changed. You can do this by writing a *Grep* search as:

Search for:

(Figure)(\d+)(-)(\d+)

Replaced with:

\1\2\.\4

This is keeping the first, second, and fourth sets of parenthesis while removing the third set, which is the en-dash and replacing it with a literal period. The \d+ is saying any number single digit or higher.

The margin note also needs adjusting. In the original chapter it was running in, where the new book has this on its own line. To do this, the search will be:

Search for:

(\@ch_mn:)(<\@ch_mnkt>)(.+?)(<\f><\@\\$p>)(.+?)(\r)

Replaced with:

\@mnkt:\3\r\@mn:\5\6

The final result of this page of text will appear as follows. I put ... to cut out some of the actual text for demonstration purposes.

@h1:What is Second Party Software?

@tx1:One of the prerequisites to be successful with this book is that you have used <\n><\@kt>Xtags<\\$p> before, preferably on an actual project. Also, I feel it's necessary that you have read through the <I>Xtags Manual <\\$>by ... See Figure 9.1.

@tx:This could be seen as ... Xtags.

@mnkt:Xtags

@mn:A valuable QuarkXPress XTension that works with the code underneath that you cannot visibly see. It works for both text and images.


@fgc:<\@fgn>Figure 9.1<\\$p><\f><\f>Leisure time for couples is spent more frequently using laptops.

The chapter now is taking the form of the new book. While all these steps may make you think that you could have achieved this faster by just manually doing it, you'll find that if you have 7 more chapters that need adjusting

that the hard part is finished. The paging part will be left. You can also add Autopage codes to the text easily as well using Grep searches. As seen below, we see the original version of the chapter opening spread, and the new version of how it matches the original book.

Can Second Party Software be the Best Investment?

chapter 9



Objectives

- Understand the relationship between computer software and how it impacts our daily lives from commuting to leisure time
- The differences between software and hardware
- How the Internet and other forms of communication such as Instant Messaging has changed the way we communicate internally within businesses
- Describe some methods companies can use to strengthen their business using the Internet and email
- Appreciate the important ways in which we look for new ways to take software to quicken tasks by using scripts and other automation methods

Can Second Party Software be the Best Investment? 141

Intro to Second Party Xtensions

Second party Xtension have revolutionized QuarkXPress and may be a big sell in the future for InDesign. I always had a good grip on QuarkXPress, but when I needed that boost of power, I knew there was a developer out there who had just the thing I needed. I would go out of my way to learn this software. I also want to help people who sometimes get a little confused on where to start. It's one thing to set up a book in Autopage, but it's another to get the Xtags, Headers & Contents, XML, and even the QuarkXPress document set up properly. The more you have working correctly up front, the less chance there will be of unforeseen problems later in the project.

What is Second Party Software?

One of the prerequisites to be successful with this book is that you have used Xtags before, preferably on an actual project. Also, I feel it's necessary that you have read through the Xtag Manual by Elm Software, Inc., which is an invaluable reference guide. It's obvious to me that there would not be a need for this book if I didn't have a distinguishing approach to the material. Where I feel this book is unique is through my step-by-step approach, helping you achieve the task without having to keep referring back to certain sections to pull it off. I always have learned through instruction like this and I always retain something once I do it. It sticks in the memory better than just reading about descriptions of various tasks. See Figure 9-1.

This could be seen as a justifiable excuse to some degree, especially when it is the busiest time of the production season and you're in the middle of several books. It is sometimes difficult to put aside the necessary time to focus and complete the up-front tasks. The up-front time spent on coding the sample would have been saved throughout the project. In my opinion, a successful, cost-effective project relies heavily on the up-front setup of Xtags.

This book has not covered every aspect of Autopage because that is not my intention. My main objective is to show the user how to get more out of the program by approaching the program in a different manner than the user might be used to. After following through the book and learning how to do all of the procedures, Chapter 9 is full of step-by-step examples that will give you an opportunity to apply the various functions of the program. I feel if intermediate and advanced users work through these, they'll find themselves more proficient than before. Many of these concepts you'll already know, but my hope is that you'll learn some new and important things along the way.

The Impact on Profitability

Since I've been using the program, I've noticed there are many misconceptions about Autopage and its uses. It can be used in many areas of publishing, but does it





FIGURE 9-1 Leisure time for couples is spent more frequently using laptops.

Original Look

9 Can Second Party Software be the Best Investment



Chapter Objectives

After studying this chapter you will be able to:

1. Understand the relationship between computer software and how it impacts our daily lives from commuting to leisure time
2. The differences between software and hardware
3. How the Internet and other forms of communication such as Instant Messaging has changed the way we communicate internally within businesses
4. Describe some methods companies can use to strengthen their business using the Internet and email
5. Appreciate the important ways in which we look for new ways to take software to quicken tasks by using scripts and other automation methods

The Impact of Computer Software on Our Lives 143

Intro to Second Party Xtensions

Second party Xtension have revolutionized QuarkXPress and may be a big sell in the future for InDesign. I always had a good grip on QuarkXPress, but when I needed that boost of power, I knew there was a developer out there who had just the thing I needed. I would go out of my way to learn this software. I also want to help people who sometimes get a little confused on where to start. It's one thing to set up a book in Autopage, but it's another to get the Xtags, Headers & Contents, XML, and even the QuarkXPress document set up properly. The more you have working correctly up front, the less chance there will be of unforeseen problems later in the project.

What is Second Party Software?

One of the prerequisites to be successful with this book is that you have used Xtags before, preferably on an actual project. Also, I feel it's necessary that you have read through the Xtag Manual by Elm Software, Inc., which is an invaluable reference guide. It's obvious to me that there would not be a need for this book if I didn't have a distinguishing approach to the material. Where I feel this book is unique is through my step-by-step approach, helping you achieve the task without having to keep referring back to certain sections to pull it off. I always have learned through instruction like this and I always retain something once I do it. It sticks in the memory better than just reading about descriptions of various tasks. See Figure 9-1.

This could be seen as a justifiable excuse to some degree, especially when it is the busiest time of the production season and you're in the middle of several books. It is sometimes difficult to put aside the necessary time to focus and complete the up-front tasks. The up-front time spent on coding the sample would have been saved throughout the project. In my opinion, a successful, cost-effective project relies heavily on the up-front setup of Xtags.

This book has not covered every aspect of Autopage because that is not my intention. My main objective is to show the user how to get more out of the program by approaching the program in a different manner than the user might be used to. After following through the book and learning how to do all of the procedures, Chapter 9 is full of step-by-step examples that will give you an opportunity to apply the various functions of the program. I feel if intermediate and advanced users work through these, they'll find themselves more proficient than before. Many of these concepts you'll already know, but my hope is that you'll learn some new and important things along the way.

The Impact on Profitability

Since I've been using the program, I've noticed there are many misconceptions about Autopage and its uses. How far you go with the Xtension is up to you. You can begin using the software the first day and gain more knowledge with each book, or you can explore functions whenever you have time and amuse others in your findings.



FIGURE 9-1 Leisure time for couples is spent more frequently using laptops.

New formatted chapter that matches the style of the book.

Adding Autopage Codes with “Grep”

If you are planning on using Autopage and repaging the chapter, during the Xtags conversion stage of the project, you can add the Autopage tags fairly easy using Xtags and a *Grep* replacement.

In this example, any occurrence of “@bib_ttl” will be changed to begin with “[LC 1 M=40p A=3]” and the following line will need to begin with “[LC 2 M=19p A=4]”.

To apply this to the following three paragraphs, you DO NOT want the “LC 2” to end up on each @bib line.

```
@bib_ttl:Bibliography
@bib:Michael Johnson, ...
@bib:Frank Wilson ....
```

To get the proper results, we must use complex “subpatterns.” To apply this requires having the “LC 2” only with the first “@bib”. There is more than one way to do this, but the way requiring the least amount of work is have it within the same line of code as the “LC 1”.

To achieve this, the *Search For* and *Replace With* windows of the *Find & Replace* will need to have this exactly:

Search for:

```
(\\@bib_ttl:)(.+?\\r)(\\bib:)
```

Replaced with:

```
\\1\\[\\[LC 1 M\\=40p A\\=3]\\]2\\3\\[\\[LC 2 M\\=19p A\\=4]\\]
```

The end result in the input file will be:

```
@bib_ttl:[LC 1 M=40p A=3]Bibliography
@bib:[LC 2 M=19p A=4]Michael Johnson, ...
@bib:Frank Wilson ....
```

The subpatterns make this possible by using the return (\\r) as a way to put the “[LC 2...]” into the next line, making this very workable for any situation including straddle heads.

The *Grep* feature is worth the price of the BBEdit alone. Once you start applying the power of *Grep*, your workflow will become more efficient and the need for a markup department to input sensitive Autopage and Xtags codes may be a thing of the past.

To save a *Grep* pattern that can be applied throughout many projects, go to the upper right of the *Find & Replace* window, select the *Patterns* pull down menu, and choose “Add”. An *Add/Replace Grep Pattern* dialog will appear to name the pattern.

The same works for Figures and Tables. For example, in the text flow you will have a reference for a figure. You can find the callout such as:

see Figure 1.1.

Then you can write a Grep search that finds this callout and puts in the Autopage code:

Search for:

```
(\@[a-z])([a-f]|[h-z])(\w+:)(.+(Figure \d+\.)(\d+)(\r
```

Replaced with:

```
\1\2\3\4\5\[AR \6]\6\7
```

This makes sure that on any line (other than one starting with style @fg) that it looks for references for figures. The style name for the figures is @fgc, so I do not want it to match on those lines. The only real issue that can happen here is that if there are multiple references to the same piece, a reference will be put in for each. You can then write a script that will go through and remove duplicate references.

Another example is in the margin notes. If you know the margin note will fall with the key term, this can be fairly easy to do. With Autopage, the identifier can be anything, so you can just name it the same as the first word or two words in the key term. For example:

The onset of <@kt>wireless Internet<@\$p> has been important.
Using <@kt>laptop<@\$p> computers...

You would do a *Grep* replacement of:

Search for:

```
(<@kt>)(\w+)( |)(\w+|)(<\@$p>|.+(?<\@$p>)
```

Replaced with:

```
\1[[SR \2\4]]\2\3\4\5
```

The end result would look like this:

The onset of <@kt>[[SR wirelessInternet]]wireless Internet<@\$p> has been important. Using <@kt>[[SR laptop]]laptop<@\$p> computers...

This would also work with margin notes by putting the [[S laptop]] code using the character style of the <@mnkt> up to the <@\$p>.

This is a small sampling of how the Autopage codes work, but it gives you a basic idea where to begin. I've saved hours on a project by just having this figured out at the beginning of the project and, other than a few duplicate code errors, the chapters paged very well.

Altering Boxes to New Style

Matching the look of boxes is very important and really requires using the *Grep* replacements to execute. What seems to happen in many cases is that new styles appear in the book that you need to change that aren't in the destination book. This requires creating the new styles. Sometimes the boxes just need to have the look changed. Here is an example of what the box is in the original book.

WIRELESS INTERNET

With the onset of online shopping, auction companies, and the ease of use through wireless Internet, home business ventures are growing to be where the big money lies.

Unlike the commercials that you see where it shows the entrepreneur sitting by his pool sipping a mixed drink, a home business does require a lot of extra time. Don't fall victim to how people make you believe that being in business for yourself will allow you freedom that being in a work setting hinders. Chances are you'll be working for a company that has a 7-5 work day and you'll be expected to answer the phones and be available through those hours, and many times, even later.



This box is very basic and only really has the image to deal with, which is very easy to move into a different box and position using a new Xtags string. The major difference in this box lies in the fonts and the background art that is in the destination box. Here's how it will need to look:

Wireless Internet

With the onset of online shopping, auction companies, and the ease of use through wireless Internet, home business ventures are growing to be where the big money lies.

Unlike the commercials that you see where it shows the entrepreneur sitting by his pool sipping a mixed drink, a home business does require a lot of extra time. Don't fall victim to how people make you believe that being in business for yourself will automatically allow you freedom that being in a work setting hinders. Chances are you'll be working for a company that has a 7-5 work day and you'll be expected to answer the phones and be available through those hours, and many times, even later.



The secret here is to first export the original box and then figure out what the new box will need to be. For the original box, the Xtags strings are:

```
<&pbu2(189,14,142,107,,,,,(n),(,100),,n,,(9,14,7,6),m,,,,,
,"56Disk:Chapter 06:bx1.eps",,,)><&tbu2(0,0,348,165.989,,,
n,0.5,(n),(,100),,K,30,,,,(12,9,6,9),,,, ,)>@bxt:Wireless Internet
@bx1f:Box text first paragraph
@bx1:Box text 2nd paragraph<&te><&g(2,1)>
```

By looking at this, it is apparent that there will be quite a few things that need to be taken care of. The style names will change, the Xtags fields will change, but these can be removed and made into small macros that need to be in the translation table. That will be the easiest method here.

Here is how the destination box will need to be coded for the same box content:

```
<&pbu2(189,14,142,107,,,,,(n),(,100),,n,,(9,14,7,6),m,,,,,"56 GB
Disk:Xtags Book:Images:Chapter 06:fg06_004.eps",,,)><&tbu2(0,0,29p,
35p?1p9,,,k,n,0.5,(n),(,100),,n,30,,,,(12,12,9,12),,,,,)>@bx2t:<Bt-
10k-88f"Optima"> <k0>|<Bf"StoneSans-Bold"> <t0>Wireless Internet
@bx2f:Box text first paragraph
@bx2:Box text 2nd paragraph<&te><&pbu2(0,0,29p,11p,,,k,n,,(n),
(,100),,n,,,m,,,,,"56Disk:Chapter 06:fg01_005.eps",,,)><&g(1,2,3)>
```

Where we can save time here is by adding most of these tags with a translation table. That way the replacement will be easier. Here is how the tags can appear in the text file:

```
[[b1]]Chapter 06:fg06_004.eps[[b2]]@bx2t: [[line]]Wireless Internet
@bx2f:Box text first paragraph
@bx2:Box text second paragraph[[b3]]
```

How these replacements will need to be written are as follows:

Search for:

```
(<\&pbu2\(\d+,\d+,.+?\(9,14,7,6\).+?56Disk:)
```

Replaced with:

```
\[[b1]]
```

The [[b2]] translation table entry will need to be replaced with:

Search for:

```
(\",,,\\)><\&tbu2\((0,0.+?)\\@bxt:)
```

Replaced with:

```
\[[b2]]\@bx2t:
```

The `[[b3]]` translation table entry will need to be replaced with:

Search for:

`(<\&te><\&g\2,1\>`

Replaced with:

`\[\b3]`

The `[[line]]` translation table entry will need to be replaced with:

Search for:

`(\@bxt:)`

Replaced with:

`\1\[\line]`

The actual translation table entries will appear as:

```
[[b1]]      <&pbu2(189,14,142,107,,,,,(n),(,100),,n,,(9,14,7,6),m,,
,,,,,"56 GB Disk:Xtags Book:Images:
[[b2]]      ",,)><&tbu2(0,0,29p, 35p?1p9,,,k,n,0.5,(n),(,100),,n,30,,
,,(12,12,9,12),,,,,)>
[[b3]]      <&te><&pbu2(0,0,29p,11p,,,k,n,,(n), (,100),,n,,,m,,,,,
,"56Disk:Chapter 06:fg01_005.eps",,,)><&g(1,2,3)>
[[line]]    <Bt-10k-88f"Optima"> <k0>|<Bf"StoneSans-Bold"> <t0>
```

Although this may appear to be a lot of work, it really makes it easier throughout all the chapters. Each time you have to change a box, you merely need to run this against the boxes.

The only real problem I anticipate in dealing with boxes is if they weren't set up with Xtags to begin with and were put together by hand. You can get all kinds of mixed results because the pager may not have grouped them properly, did manual overrides, etc. In this case, you may just want to regroup them and write one long replacement string that can catch everything regardless of what the numbers are. This is why I always say you need to know your content. Don't just assume that everything is going to be correct during the export. Look through a chapter to see what you are dealing with first.

In Closing

Although there is so much more we could cover on the possibilities with custom publishing, this should give you a great place to begin. More of this type of work is going to be available in the future as more publishers are setting their sights on custom publishing. This is something you should really learn.

7 Xtags Working with Autopage

CHAPTER

Autopage may only be used by a percentage of the Xtags users, but it is a very important subject to discuss because it will increase your productivity more than you can imagine. I feel if you are a pager, not using Autopage in conjunction with Xtags is really missing the full spectrum of what is possible. Your goal as a paginator should be to speed up your workflow and be as cost-effective as possible.

It's a very critical time to be effective with a program like Autopage with all of the changes in the publishing field. Many U.S. companies have reduced their in-house production staffs and have been outsourcing to stateside freelancers and foreign composition houses to meet a large part of their paging, keystroking, and art needs. How this translates to a stateside company is that it is critical to find ways to automate the process to remain competitive. This is essential in the survival and longevity of any publishing company.

My goal with this chapter is to show users of Autopage or even those who might have some interest in the program where Xtags comes into play with the program. The two work very well together and so much is possible with the exporting now with the rise of XML usage, that I can't see where someone wouldn't want to use the two together.

I started using Xtags and Autopage back in 1997. Xtags was the one I figured out first and then took that knowledge and applied it further when working with Autopage. I quickly found that I needed both to be fully effective as a pager and I was one of the most productive pagers. This was not by accident. It had so much to do with how far I could take both programs. Even when I was assigned the most difficult medical or mathematics book, I was still under budget with my paging and setup.

The main focus of Autopage should be to reduce the time spent on manual pagination and for overall consistency of the page layout. This has become critical not only for staying profitable in your business, but also for staying competitive with foreign companies who can do the same work in the same time at a fraction of the price because of lower overhead and reduced labor

costs. Autopage, combined with Xtags, can give a pager the edge if used effectively and proficiently.

Autopage can be used in many areas of publishing including (but not limited to) books, catalogs, and telephone books. The further you push yourself into this program, the easier your job will become. Initially the learning curve might seem overwhelming, but the more you use the program, the easier many of the tasks will become. Every year textbook designers go beyond the previous margins of creativity, so it's very possible that the functions you learn about today with Xtags and Autopage will be necessary features in an upcoming project.

I'm going to touch on some different areas of Autopage that can be enhanced by using Xtags in combination. While non-users may get a little lost here, you can at least see some advanced features of the program and how Autopage and Xtags works together.

The Paginate Window in Autopage

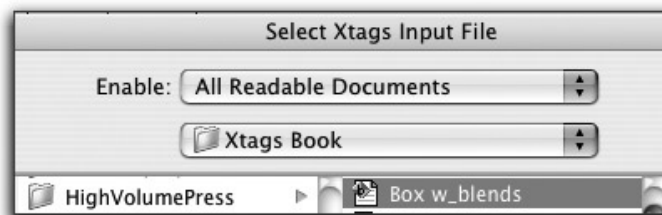
The *Paginate* window in Autopage, found in QuarkXPress under the pull down menus at the top of the screen, controls the conversion, paging, and exporting of the document.



Within this menu there are a variety of buttons that can be selected. Autopage 5.8 and 6.0 differ slightly in the numbering. You'll see here that you can use this window to do the *Get Text with Xtags* instead of doing it under File.

I habitually use *File: Get Text with Xtags* to import my text before I go to the Paginate window, but to use the Import Xtags (☒ 1), here are the steps.

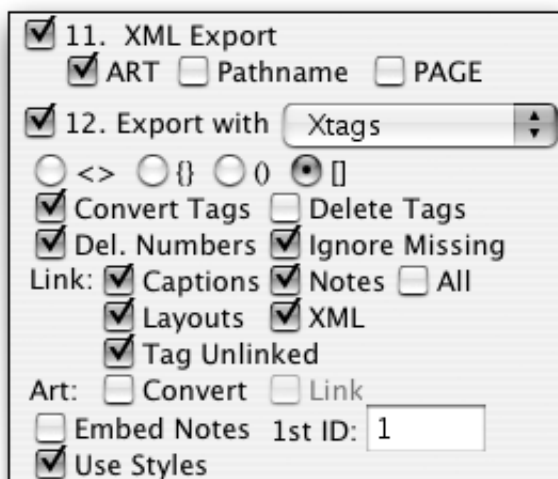
1. Select the Import Xtags ☒ 1 and select *Proceed*.
2. A window appears to *Select Xtags Input File*. It lets you navigate through the folders. Then select the file you want.



3. The file will import. If there are problems, the Xtags error message will appear.
4. Xtags will finish and then if other selections are made such as ☒ 3 XML Tags or ☒ 4 Running Head tags, Autopage will move on to these.

Repurposing Content

If you are planning on repurposing your Quark content for multimedia, it is very important to use Xtags and XMLxt. The *Paginate* window and the *Batch Export* option have the ability to export the tags with many options for both XML and for Xtags exporting:



☒ 11 is for XML exporting and ☒ 12 exports the document (including all art, text, layout changes, floating elements, etc.) into Xtags or XPress Tags. The Xtags export is more specific, so this is another added bonus to owning the program. When I've done roundtripping where I've taken regular tagged Quark files and converted them to finished XML products, the Xtags export made the content in a workable format where everything came together. All text boxes, floating elements, etc. were all useable.

Then I decided what I needed, could delete what was unnecessary, find the picture and text boxes (&tbu2, &pbu2, etc.) and multipurpose that content into XML. The main use currently with ☒ 11 and ☒ 12 is for an XML workflow using the XMLxt XTension. If you have a workflow that supports XMLxt, you have probably used this at some stage. The thing about this is that you can get any text that sits in the text area of the entire document to export so that it can be used later.

I will go into more detail concerning this in Chapter 8. This is a very powerful exporting text tool.

Side Art – Using Reference

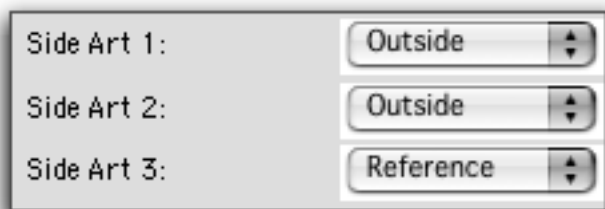
I've paged some extremely difficult books using Autopage that some said couldn't be done. I'm not declaring that they were easy, because they were just the opposite, but when someone says it can't be done I give it my best shot to prove them wrong. I try to argue that it should be paged in Autopage unless there isn't a main text flow, where each page is unique rather than following a set design. Here is an example of a layout with annotations:

I am going to have to change my present job, or I am going to be fired. Yesterday, I did
^{poorly}
 poor on my clean up procedures. I had worked 12 hours at my job and hadn't eaten anything
^{of} ^{have}
 most off the day. Yesterday, when I thought I would get some downtime to get a meal, the boss
^{over}
 made me take offer for an absent employee. So I was stuck at the front desk, taking care of
^{very}
 business. That job is becoming vary tough.

English books with heavy use of annotations have been argued to be too complex for Autopage. I'm here to tell you this opinion isn't true. Annotations can be achieved by combining the power of Xtags and Autopage, but the set up can be perplexing.

The breakdown to achieve this is as follows. You'll need to be using an Xtags translation table to accomplish this. I'm mainly showing how to position annotated words above the text in this example.

1. In the *Horizontal Alignment* parameters, make sure the "Side Art 3" is set as "Reference".



Unfortunately, the *Horizontal Alignment* parameter is the only place to apply this feature. At this point, I do not know of a code equivalent to call in "Reference" for an override.

2. Find an average size for your side element text box. For example, if it is by 7p, then have your ANNO style center or left justify above the word.

3. Have an Xtags code `[[SA]]` before the ANNO and a `[[SA2]]` and `[[SA3]]` that go around the Autopage “Object” tag. The reference tag will immediately follow next to the word the ANNO will be placed above.
4. Within the `[[SA2]]` code you’ll have a soft return.
5. The `[[SA]]` code calls out the style sheet for the annotation. This way it can be typed directly before the word. The tagging will look like this:

`@T1:She is [[SA]]thinking[[SA2]] [[S 2]] [[SA3]] [[SR 2 V=3]]thankin`

6. The SA and SA2 will be set up in Xtags as:

`[[SA]] <&tbu2(0 B,0,7p,2p3,,,n,)@ANNO: Starting the Box`

`[[SA2]] <\n><@$p> Soft returning
AutoTag`

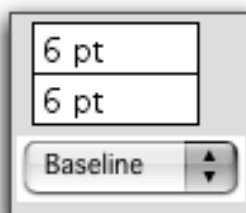
The AutoTag will be soft returned and will pick up the same style as the text so it will line up properly

`[[SA3]] <&te> Closing the text`

This may seem convoluted, but it will save time. It may seem very complex to get someone to code these properly. If explained, the coding department should have no problem. If using a translator, you can code this in. Using MacPerl can make this come together very easily. Writing an AppleScript can also be achieved to get this result.

Regardless of which method is used, it’s critical that the coder types the ANNO term right before the term or word it is going to be placed above once paginated. This makes all the difference when using this method. Any deviation will cause errors.

7. The *Side Element Placement* parameter will use “Art 3” and will need to be set up with 6 pts in the “Side Element Offset from Margin” and “Baseline” for the vertical alignment.



8. Since the side art will always need to be left, in the *Horizontal Alignment*, place the “Side Art 3” text alignment as “Left”. This can also be accomplished by putting a `<*L>` in the Xtags.

Rotated Tables

Rotated tables are one of the easiest items to work with in Autopage, but these require being set ahead of running the AutoTag conversion or with Xtags. Autopage only reads unrotated tags, so it's very important that the tag be unrotated, although the remainder of the table can be set at any angle.

Add Tag Here

Table 1.1 Test Results for Men and Women:											
Males:						Females:					
19-30	5717	395	17.5	109	109	19-30	5717	395	17.5	109	109
31-50	5559	307	17.9	97	99	31-50	5559	307	17.9	97	99
51-70	5165	565	17.5	73	67	51-70	5165	565	17.5	73	67
71-90	1751	530	17.5	67	75	71-90	1751	530	17.5	67	75
90-95	1791	539	15.7	65	63	90-95	1791	539	15.7	65	63
96-100	1699	515	13.7	63	65	96-100	1699	515	13.7	63	65
100+	1536	195	19.9	56	65	100+	1536	195	19.9	56	65

For a rotated table:

1. Bring in an unrotated text box and group this box on top of the existing rotated table.
`<&tbu2(95 B,95,61,25,90,,,n,,(,n),(,100),,n,,,,,,,,,)>@tbl:[[T 1]]<&te>`
2. The table will eventually need to be rotated 90°, but for now leave it unrotated so you can set your tabs, adjust, etc. The shrink-to-fit doesn't work well with the rotation, so this is the best way to start. The x and y paste-

board positioning will be at 80 x so when rotated, there is room for the rotation. Here's an example of the table code:

```
<&tbu2(80 B,80,30p,45p?,,,n,0.5,(n),(,100),,K,10,,, (7,7,7,4),,,,,)>
```

3. The [[T ID#]] object tag will be in the small rotated text box. After you have adjusted the tag, rotate the text.
4. The table will now page properly. Here is the full code:

```
<&tbu2(95 B,95,41,25,-90,,,n,,(n),(,100),,n,,,,,,>@tbl:[[T 1]]<&te>
<&tbu2(80 B,80,30p,25p?,,,k,n,0.5,(n),(,100),,K,10,,, (7,7,7,4),,,
,,,>@tbl:all of the table body text<&te><&g(1,2)>
```

Left and Right Art Placement Option

The left and right table placement option opens up practically endless possibilities when it comes to working with boxes and tables that have differences on the verso and recto pages. What few are aware of is that this will also work with figures as well using the [[A]] and [[AR]] codes. I ran across this when I was experimenting.

Initially, you might not see the benefit of this, but I assure you there are times when this is needed. The *Horizontal Alignment*, for the most part, can handle moving the captions for most pieces, allowing differences on recto and verso pages. If art has a grouping of three elements, this will not work with the *Horizontal Alignment*, thereby making it necessary for using the left and right art placement. Here's an example with a rule running the full width of the image.

Waves provide entertainment
at Daytona Beach, Florida




The bar will not allow Autopage to flip the captions to the outside as needed. Some pagers may find it easier to just add the bar after paging, which will work, but that seems to be defeating the idea of automation. This can be handled two different ways, but both will require extra work and the use of Xtags.

Solution 1: Using Xtags with Left/Right Art Placement

When the L and R table placement was developed, I noticed afterward that this would also work for art. Therefore, this solution is to import both pieces two different ways through Xtags. The concept here is to have the left piece import with the caption on the outside and the rule above both the caption and art. The right page art will import with the caption to the right, with the rule running full length regardless of the length of the art.

On the verso page it is easy to get this result, but on the recto page, I achieved this by having the rule read off the right edge of the caption and run -42p in reverse. This allows the piece of art to shrink-to-fit, but doesn't have any impact on the rule running the full length above the two items.

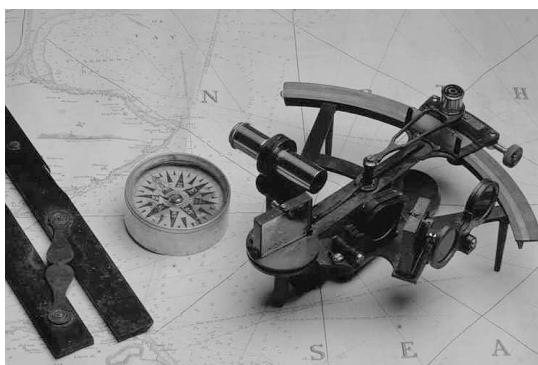
To achieve this, follow these steps:

1. Calculate the length of the rule, length of caption, and maximum width and depth the art can be.
2. Put the piece of art on the 0x, 0y coordinates of the page. It's easier to calculate the Xtags applying this method.
3. Select the "Item" tool  and group together. Select *Edit:Copy Xtags* unless you prefer writing the codes out.
4. In a text box, place the cursor, and select *Edit:Paste*. The tags will contain all the pertinent information; however, you will have to change the items needing TR1, BL1, and so on.
5. For this example, here are the two different Xtags strings:

RECTO PAGE:

```
<&pbu2(15p B,1p5,29p?-.75,40p?-.75,,,,n,,(n),(,100),"Solid",
K,0,,m,,,,,,"nautical.tif",,)><&tbu2(12 TR1,0,10p,8p?)>
@CAP:Nautical Equipment<&te><&pbu2(-42p TR1,-12,42p,5,
,,n,,(n),(,100),"Solid",K,50,,m,,,,,,"",,)><&g(3,2,1)>
```

Here's a reduced version of how the recto image will import.



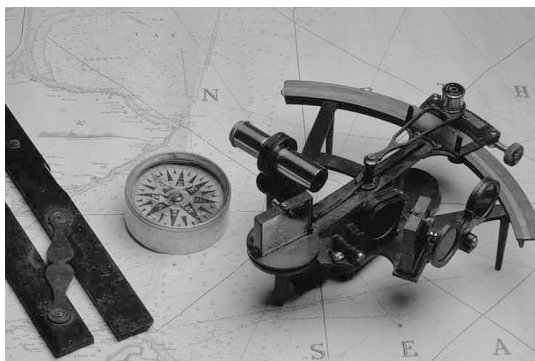
Nautical Equipment

VERSO PAGE:

```
<&tbu2(1p B,1p5,10p,8p?)>@CAP:Nautical Equipment<&te>
<&pbu2(12 TR1,0,29p?-.75,40p?-.75,,,,n,,(n),(,100),"Solid",K,0,,m,
,,,,,"nautical.tif",,)><&pbu2(0p TL2,-12,42p,5,,,,n,,(n),(,100),
"Solid",K,70,,m,,,,,"",,)><&g(3,2,1)>
```

Here's a reduced version of how the verso image will import.

Nautical Equipment



6. The Xtags translation table would have to be broken up to look like this:

Recto Page:

```
[[f1]] <&pbu2(15p B,1p5,29p?-.75,40p?-.75,,,,n,,(n),(,100),"Solid",K,0,,m,,,,,"
[[f2]] ",,)><&tbu2(12 TR1,0,10p,8p?)>
[[f3]] <&te><&pbu2(-42p TR1,-12,42p,5,,,,n,,(n),(,100),K,50,,m,,,,,"",,)>
<&g(3,2,1)>
```

Verso Page:

```
[[f4]] <&tbu2(1p B,1p5,10p,8p?)>
[[f5]] <&te><&pbu2(12 TR1,0,29p?-.75,40p?-.75,,,,n,,(n),(,100),"Solid",K,0,,m,,
,,,,"
[[f6]] ",,)><&pbu2(0p TL2,-12,42p,5,,,,n,,(n),(,100),K,70,,m,,,,,"",,)>
<&g(3,2,1)>
```

7. The coded files would look like this:

```
@CAP:[[f1]]nautic.tif[[f2]]@CAP:Nautical Equipment[[f3]]
@CAP:[[f4]]@CAP:Nautical Equipment[[f5]]nautic.tif[[f6]]
```

8. After this is set up, the next step is importing the text into the Quark file.

9. Locate each left and right piece of art on the pasteboard. Move each piece into the library, the left piece marked “FIG1_L”, the right piece marked “FIG1_R”, consecutively.
10. Code the reference as `[[AR 1 L=FIG1_L R=FIG1_R]]`.

When Autopage runs, it will place the left piece on the verso page, or the right piece on the recto page, depending on which page the reference lands on. If you prefer, you can also import without the text in the caption and use the “T=E” method.

This process, although lengthy, saves adjustments during paging such as putting a rule above each piece, repositioning, and so on. Just for the record, this entire example, including typing the explanation, only took 50 minutes. Throughout an entire project, this process saves time and headaches once it's perfected.

The best solution for the library is to duplicate the standard library for the book and have one for each chapter. The time it takes to put them in the library and name the piece is minimal in comparison to the time you'll save.

Solution 2: Manually Positioning the Rule

The second possibility requires the pager to manually place the rule with the art, but without having to adjust the placement of the figure. This is a better solution for those less advanced with Xtags. If you were to pull the figure in the standard way, you would need to move the figure 1p1 down from the top to make room for the rule. This would cause reworking during pagination.



Construction is a constant process with year-round employment.

My idea here is to offset the figure, using 1p1 of white space when pulling it in with Xtags and having the caption sit 1p1 from the top of the image. This will place the art properly with all of the allotted space for the rule that needs to be manually placed. To do this:

1. When setting up the Xtags picture placement, the *First Baseline* offset will need to be 1p1:

```
<&pbu2(15p B,0,29p?-.75,40p?-.75,,,,n,,(n),(,100),"Solid",K,0,,m,,,1p1,
,,"construction.tif",,)><&tbu2(12 TR1,1p1,10p,8p?)>@CAP:Construction
is a constant process with year-round employment.<&te><&g(1,2)>
```

The extra white space in the figure will not cause any issues when printed, during FlightCheck procedures, and so on. Here's how it will import. The rule around the image is only for demonstration purposes to show the offset.

2. To do this effectively, the "Horiz. Art Caption 2" will need to be set "Outside" in the *Horizontal Alignment* parameter. To override the parameters, the code will need to read [[AR 1 X=O]], whichever you prefer. My best advice if using the parameters for the outside placement is to use V=2 so other art will not be affected by the outside placement.
3. I suggest setting the placement to I=T (float top) for all these pieces when first coded. This forces the art to sit at the top of the page so the bar can snap in place without having to position with the x and y coordinates.
4. The ideal coding structure for these pieces will be:

```
[[AR 1 I=T X=O]]
```

5. After pagination, group the art and caption with the bar. Here's the same image with the bar included:



Construction is a constant process with year-round employment.

This process achieves the results without working with the palette. Don't group the bar/rule with the figure until paging is finished.

Once the bar is part of the images, the captions will no longer go to the outside during pagination. This method reserves space for this function.

Text Wraps (Runarounds in Autopage)

Text wraps are heavily implemented in complicated designs. It's something that can give a title more content while conserving space. There are a couple of ways to handle wrapping images using Autopage. At this time, Autopage doesn't support wraps; however, it's a critical function of paging, so working around it is in your best interest. There are plans for this in the near future.

I've heard paggers declare "this book cannot be paged using Autopage because of the wraps." I've yet to do a book with wraps that I couldn't use Autopage. Sometimes it requires starting and stopping quite often, but the project was still completed in a timely manner. Of the methods I use, one requires minimal reworking that bypasses the error message associated with runarounds. Here's an example of a design with wrapped images.

Method 1: Treating Wrapping Art as Side Art

This first scenario is to treat the wrapping art as side art. Generally, I've found designers and editors are pretty consistent when having art wrap. They usually have two or three sizes of photos that have the ability to bleed. Many times they are vertical photos with a width between 12p and 15p. The horizontal photos are between 18p and 24p. Although the depth is an uncertainty, this makes calculating the amount of space the photos will take up when the runaround is applied much easier.

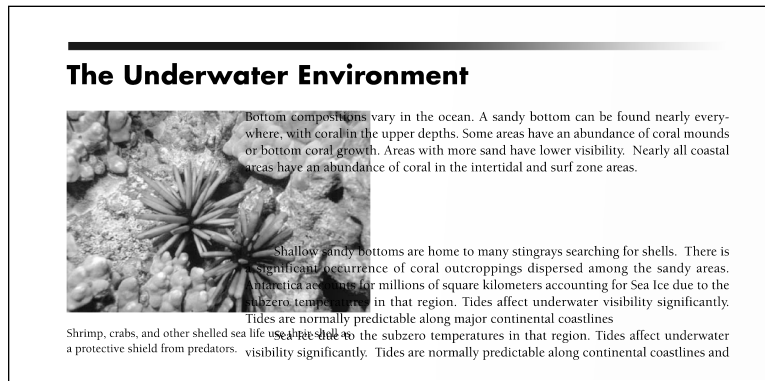
1. I leave the runaround off when paging, having the art fall to the outside using side art. The intention here is that the widths are uniform on a few images. I noticed that the average width of the photos is 18p. The margin is by 9p with 1p in the gutter. That leaves 8p that will need to underlay. I set the *Horizontal Alignment* to "Outside" for "Side Art 1" and the *Side Art Placement* parameter as shown below:

Allow Side Element To Straddle Margin:	<input checked="" type="checkbox"/>
Treat Side Element as Underlay:	<input checked="" type="checkbox"/>
Side Element Offset from Margin (left page):	-8p
Side Element Offset from Margin (right page):	-8p

2. I combine the power of Xtags to give me an edge. In the paragraph where the art is referenced, I add extra space to the paragraph that I feel the wrap will add to the text in an Xtags code. For example, once I turn the runaround on, the art will most likely push the text 3 or 4 lines for horizontal photos and 4 or 5 lines for vertical photos from the original position before the runaround is applied. I add this Xtags string:

<*p(,,,,,48,,)>

This adds 48 points to any style sheet the reference falls within. There will now be 48 points of white space below the paragraph containing the photo reference when the image is first pulled in. Here's an example. At first when paged, the image is in the background without the runaround on. The text is flowing over the 8p underlayed portion of the image.



3. This can be inserted in the input ASCII file two different ways:
 - a. It can be added as a macro by the markup department with something simple like an `[[sp]]` that is part of the translation table. The ASCII markup will be added as:

```
[[AR P#]][[sp]]
```

The translation table will contain:

```
[[sp]] <*p(,,,,,48,,)>
```

- b. It could also be added using MacPerl, or through an AppleScript. For MacPerl, this would be entered:

```
s/(\[AR P\](\d+)(\))/s1$2$3\[sp\]/g;
```

or I could add the code in MacPerl itself and avoid the translation table altogether by coding:

```
$sp = "<*p(,,,,,48,,)>";
```

```
s/(\[AR P\](\d+)(\))/s1$2$3$sp/g;
```

4. Once the runaround is applied, the text will probably move forward 4 lines. Since the extra space has been added after the referenced paragraph, manually go into the paragraph and remove the extra space. Your page should break the same as it did prior to the runaround being applied. In this case the second paragraph ends in the exact location it did with the extra space applied. I find it achieves this result about 75% of the time.

The Underwater Environment



Shrimp, crabs, and other shelled sea life use their shell as a protective shield from predators.

Bottom compositions vary in the ocean. A sandy bottom can be found nearly everywhere, with coral in the upper depths. Some areas have an abundance of coral mounds or bottom coral growth. Areas with more sand have lower visibility. Nearly all coastal areas have an abundance of coral in the intertidal and surf zone areas.

Shallow sandy bottoms are home to many stingrays searching for shells. There is a significant occurrence of coral outcroppings dispersed among the sandy areas. Antarctica accounts for millions of square kilometers accounting for Sea Ice due to the subzero temperatures in that region. Tides affect underwater visibility significantly. Tides are normally predictable along major continental coastlines.

Sea ice due to the subzero temperatures in that region. Tides affect underwater visibility significantly. Tides are normally predictable along continental coastlines and within the

The advantage to this method is that you will most likely have less adjusting to do on the following pages because the page should have fallen correctly or close to the desired result.

The only real disadvantage can be if the referenced paragraph is the last paragraph on the page. The 4p will not help your page because the space is in the style, but isn't doing anything because there's no subsequent text.

Method 2: Using an Anchored Box

Using an anchored text box (*Ascent*) at the beginning of the paragraph to add space for the runaround is another acceptable method. I don't use this method too often because I have had great success with the previous scenario. To achieve this:

1. At the callout, it's mandatory to put an anchored picture box as a placeholder for the runaround.
2. To get this to anchor can be handled two different ways:
 - a. Have the markup department key in an Xtags code `[[anch]]` that would be match the translation table. The code would be to this effect:

```
[[anch]]      <&pb(8p,10p,a,,(n),(,100),,n,,(1,1,2,12),m,,,,, "",,)>
```

- b. The other option would be to write a MacPerl script or AppleScript to position the macro appearing in the translation table into the text flow at the start of the style sheet each time a photo is called out. The MacPerl script would be written as follows:

```
s/^(^@.+?:)(.+?\[ \[AR P\] \(\d+\) \(\) \) /$1\[ \[anch\] \]$2$3$4/g;
```

If the text in the paragraph was coded as:

```
@T1:The example is shown in Photo [[AR P2]]2.
```

the MacPerl script would change this to:

```
@T1:[[anch]]The example is shown in Photo [[AR P2]]2.
```

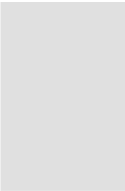
This ensures that the anchored piece would import each time at the start of the paragraph containing the photo reference. I personally would also include the reference tag directly after the `[[anch]]` to ensure that the photo lined at the top of the anchored box. I didn't want to overcomplicate this, but that would eliminate the placement issue.

3. When this imports with Xtags, the text will have the anchored picture box as a placeholder for the area. Here is an Xtags code for the anchored picture box placeholder:

```
<&pb(78,118,a,,(n),(,100)),n,12,(,2,12),m,,,,,"",,)>
```

This way you could still use the first step from Method 1 and get the same effect. Here's an example of the placeholder. For demonstration purposes, I put a 10% screen on the anchored placeholder for visibility.

The Receding Shoreline




Sea level will cause exposure of the coastal plain and erosion by rivers draining out to the receding shoreline. On the shelf, deep water sediments shallow water sediments. The example is shown in Photo 2[[AR P2]]. If sea level fall is extensive enough, the entire shelf will develop a sequence boundary. The shelf edge will correlate to a stratigraphic horizon in the more continuous sequence of deep water sediments. This horizon is the correlative conformity of the shoreline.

Shallow sandy bottoms are home to many stingrays searching for shells. There is a significant occurrence of coral outcroppings dispersed among the sandy areas. Antarctica accounts for millions of square kilometers accounting for Sea Ice due to the subzero temperatures in that Antarctic and polar regions. Tides affect underwater visibility significantly. Tides are normally predictable along major continental coastlines

Sea Ice due to the subzero temperatures in that region. Tides affect underwater visibility significantly. Tides are normally predictable along continental coastlines and within the

4. Once the paginator is run, the photo will fall into place or very close to this location as shown in the example below. There should be minimal adjustment, however, you would need to adjust the depth of the anchored box depending on the depth of the art.

The Receding Shoreline



The effects of the receding shoreline.

Sea level will cause exposure of the coastal plain and erosion by rivers draining out to the receding shoreline. On the shelf, deep water sediments shallow water sediments. The example is shown in Photo 2. If sea level fall is extensive enough, the entire shelf will develop a sequence boundary. The shelf edge will correlate to a stratigraphic horizon in the more continuous sequence of deep water sediments. This horizon is the correlative conformity of the shoreline.

Shallow sandy bottoms are home to many stingrays searching for shells. There is a significant occurrence of coral outcroppings dispersed among the sandy areas. Antarctica accounts for millions of square kilometers accounting for Sea Ice due to the subzero temperatures in that Antarctic and polar regions. Tides affect underwater visibility significantly. Tides are normally predictable along major continental coastlines

Sea Ice due to the subzero temperatures in that region. Tides affect underwater visibility significantly. Tides are normally predictable along continental coastlines and within the

The downside to doing this is that working with anchored art can make pagination a little more difficult if the deep-anchored piece falls at the bottom of the page. If this should occur, you can reduce the depth of the anchored box, and adjust as necessary once the piece is properly positioned. The upside is that you don't have to manually take out space.

Oversized Two-Column Inline Art

Inline art in a two-column layout will fall within one column unless the art is oversized. If oversized, the art will split the columns evenly, place the art, and then restart the new columns below. If you have a design that calls for the art to be slightly oversized in order for it to bleed to the outside of each column, this can be handled by using the C=O (Caption=Only Caption) art placement option.

The C=O needs to be in a text box that takes up the area you want to be positioned. This can be tricky and most people do it after it has been imported. I will demonstrate both ways.

Handling After Import

1. Place a text box over the area of the art you desire to be positioned in the text area. In this case, the box could be no larger than the 20p column width.
2. In this box put the art reference and the C=O. The C=O can be part of the reference or the object. I usually choose the object to keep track.
3. When placed, the extra portion of the image will push out either left or right, depending on the set up. This keeps the image from using two columns of space.

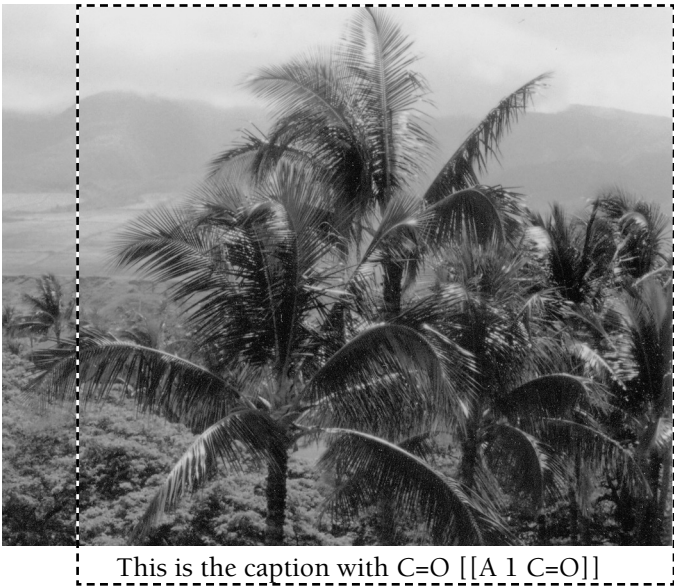
Bringing in with Xtags

To have an image import with the C=O positioned perfectly around the art using Xtags requires some real planning. You'll want the caption to be part of the text box around the portion of the image that is in the text area only. To achieve this:

1. The Xtags string is going to need relative placement vertically starting at the top left of the image extending 20p, then extending the full length of the photo with 2p extra at the end allowing for caption placement. The Xtags strings will be:


```
<&pbu2(10p B,0,30p?,46p4?,,,n,,(K,W),(100,100),,K,0,,m,,,,,"56
GB Disk:AP_Book:Art:Photos:flowers3.tif",")><&tbu2(0p TL1,
0,20p,(46p4,R,2p,1"),,,,n,,(K,W),(100,100),,n,,,,, ,b,,)>@F_CAP:
This is the caption with C=O [[A 1 C=O]]<&te><&g(1,2)>
```

2. The sample below shows the “caption only” space with a dotted line around the image. The dotted line is for demonstration only. When this places, only this portion of the image will place in the text area.



This is the caption with C=O [[A 1 C=O]]

3. When paging, you can have your horizontal art caption alignment to the inside or outside, but depending on which column it lands on, you’ll have to adjust the one that ends up between the columns. The layout would look like this when finished:

<p>In Maine, the ocean temperature rarely exceeds 65 degrees in the hottest parts of the summer. There is approximately one half a cup of salt per gallon of water on the nearby beaches. Sand Beach is nestled between mountains and rocky shores to the east of Mount Desert Island.</p> <p>Megadunes were first noticed by pilots flying over parts of East Antarctica. Satellites showed that megadunes areas were huge areas. The found one area to be the size of California. Nearly 25 percent of the Northern Hemisphere is covered by permafrost from freezing and thawing temperatures.</p>	<p>Secluded Hawaiian Beaches</p> <p>A small walk down a rocky cliff leads to Papakolea Green Sand Beach. For very private times, Pine Trees Beach is a favorite for travelers who are looking for tranquility over mass cultural appeal. Spencer Beach Park is protected by a giant reef. Other beaches worth investigating are Puako Bay, Pololu Valley Beach, Manini Beach, Kona Coast State Park and Kua Bay. All free of commercialism that surrounds Honolulu and popular beaches of that area.</p> <p>In Maine, the ocean temperature rarely exceeds 65 degrees in the hottest parts of the summer. There is approximately one half a cup of salt per gallon of water on the nearby beaches. Sand Beach is nestled between mountains and rocky shores to the east of Mount Desert Island.</p> <p><i>For very private times, Pine Trees Beach is a favorite for travelers who are looking for tranquility over mass cultural appeal.</i></p> <p>Napo'opo'o Beach Park is located at the southern end of the beautiful Kealahou Bay on a very rocky terrain. Black sand beaches are very common in the state of Hawaii. There's Moana Loa and Moana Kea and there's Hana which is part of the Maui landscape, but a trecherous road will take you to it. Not for the faint hearted, but worth the drive.</p> <p>The photos contained on these pages are not necessarily from any of the mentioned locations. For display purposes only.</p>
	
<p>This is the caption with C=O</p>	

Anchoring the H1 Side Head Instead of Side Art

Many people often have this dilemma when setting up a design. They'll have an "H1" or an "Example" head positioned in the margin and the text aligns next to it in the major column, as seen here:

Jim Carrey's Rise to Fame and Fortune

returned box office success in movies like *Notting Hill* and the academy award winning *Erin Brockovich*.

Jim Carrey is another performer who was around forever in movies like *Doing Time on Maple Drive*, *High Strung*, *Once Bitten*, and *Sudden Impact* all in support roles and never really allowed to show his skills as a comedian. Then came a little movie called *Ace Ventura* that did major things at the box office. Following that movie, Carrey has been unstoppable. Continuous blockbusters like *Liar Liar*, *Dumb and Dumber*, *The Truman Show*, and *The Mask* have proven that Carrey is at the top of his professional game.

Some may consider that when using an asymmetrical design, making a layout change full width having the left indent 11p and the first line -11p would work. If there are two lines within the title, this always presents a problem and I don't suggest this.

Making this a piece of Side Art is a bigger nightmare because if you are roundtripping your content, you can find it more of a hassle to deal with. I suggest anchoring the H1 head and using a full width layout change to accommodate the space.

1. Make a layout change for the full margin width.

```
[[LC 1 M=40p]]
```

2. The first text style sheet "TX1" will require this Xtags override.

```
<*p(84,-84,,,,g,"U.S. English")><*t(84,0,"1 ")>
```

3. Since an anchored text box will be putting in the H1 head, the style sheet will reflect the following:

```
@TX1:<*p(84,-84,,,,g,"U.S. English")><*t(84,0,"1 ")><b-4><&tb(64,50, a,,(n),(,100),,n,0,(,1,1),,,,,,)>@H1:Jim Carrey's  
Rise to Fame and Fortune<&te><b0>
```

4. The process could be streamlined in the Xtags translation table by being coded as:

```
@TX1: [[h1a]]@H1:Jim....[[h1b]]
```

Although I like using the power of Autopage whenever possible, I find this an easier scenario to work with by combining Xtags anchored text box options.

8

CHAPTER

Repurposing with Xtags

Even if repurposing your content does not fit in your current workflow, I wouldn't pass over this chapter because this is definitely the wave of the future for publishing. Some of you may think this chapter does not apply to your workflow or you will not get any benefits from it, but I suggest you reconsider. Anything else you can learn will only help you in other areas of your workflow. Maybe XML isn't what you need. Perhaps, it is just turning the document into HTML. Following similar steps, you can achieve this as well.

Repurposing is one of the more important aspects of Xtags working together with Autopage, BBEdit, and AppleScript. The entire process involves converting the formatted Quark material back to an XML format so it can be used for various platforms including e-books, web-related material, and databases. Neither Xtags, nor Autopage, will take the document to an XML state, but it will export the text, allowing you to use some type of script to achieve the results. My preferred method is using AppleScript by way of BBEdit's *Grep* in the *Find & Replace* window to produce these results.

Currently *Autopage*'s "Batch Export" converts the hidden *Autopage* tags as well as links into markup codes that can be used in either Xtags or the standard XPress tags, depending on what you select in the export window. I personally always select Xtags because there is more detailed unanchored and anchored picture box and text box information, amongst all the other paragraph and character style attributes that populate the file.

If you are using XMLxt and have a workflow already in place, you are already working in an XML workflow. However, if you are not working in an XML workflow, but are finding that you need to take paged material and have it XML ready for a customer or yourself, you may want to read this chapter on a proven method for doing this. I'm not going to go into details concerning XML because this chapter is more about how to take the paged file (without XML markup) and find a way to have it become XML markup.

I have found that any content can be exported out of Quark through Xtags via the *Autopage Batch Export* and can be produced into a workable XML file.

This is invaluable because this allows full control over the content compared to using an off-the-shelf product. XML is so specific and this allows you to customize your workflow. A few rules need to be followed in the original Quark file to save handwork:

1. All text and picture boxes in a floating element need to be grouped together. For example, all of the elements in a box need grouped. This will keep the entire box as one. If a figure inside the box is not grouped with the box, this will end up somewhere else in the text document.
2. Style sheets need to be applied on all lines of text. Do not use “No Style” or the “Normal” style sheet.
3. Each image and their caption will need to be grouped together.
4. The text flow should be continuous, whether with *Autopage* links or Quark links.
5. Content needs to be inside the margin guides. Boxes, figures, etc. cannot position outside of this area. If you have figures that are a few points above for alignment purposes, my suggestion is to duplicate the file and change the margin guides to keep the images inside. Go to the Master Pages, then *Page:Master Guides* to adjust this.
6. A marker needs to be in place for all boxes and non-referenced floating objects. For example, if you have a margin note, but there’s nothing in the text referencing this. You should put a marker like `[[m01]]` in the text next to where this appears, and one at the beginning of the margin note. This saves time later. Here’s an example of the placement:

`[[m01]]`Building Skills

Gather in a group of two to three students. One strategy for becoming better at computer skills is to actually sit down and work on things.

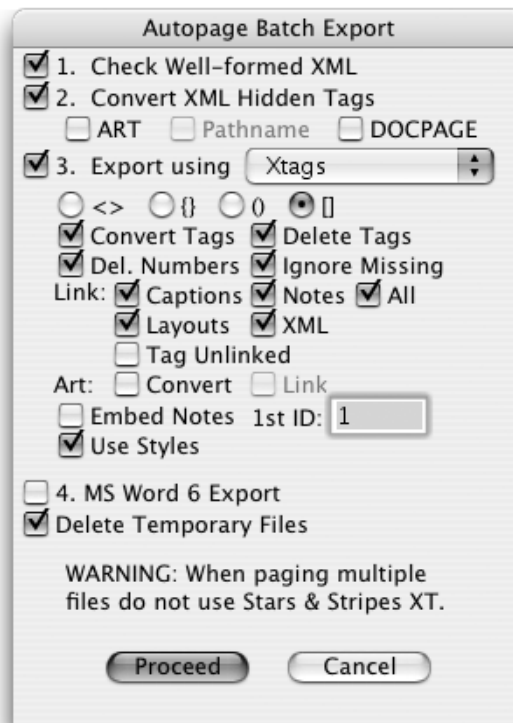
This will be explained later why this is so important, but I assure you that placing these markers should only take a few minutes (depending on the size of book).

At this point you are probably wondering where the efficiency lies. You’ll see the productivity later during the scripting stage if all of these steps are fulfilled.

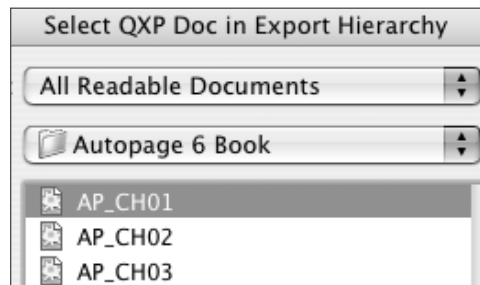
After the Quark documents have been prepared, you’ll want to put all of the chapters into a folder. You will need *Autopage* at this point in order to do the Batch Export. If you don’t have *Autopage*, this is where you would have trouble because you would have to export each text box or grouping individually and that would not be cost effective. That is one of the reasons I suggest using *Autopage*. An export engine would be a great feature for Xtags to incorporate into the program, but at this point it isn’t available.

To use Batch Export in Autopage:

1. Make sure Quark is running, but all Quark documents are closed.
2. Go to the Autopage XTension menu in the QuarkXPress menu bar and select *Batch Export*
3. Make sure the main buttons 1, 2, and 3 are selected as shown below with the same features selected. Select “Proceed”. If you don’t have any XML tags hidden in the document, the program will pass over step 1.

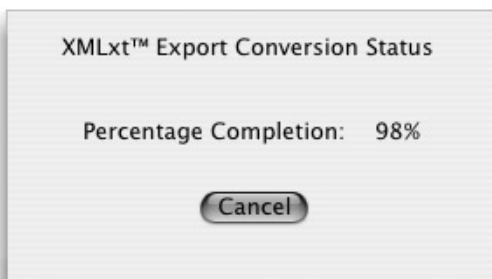


4. The dialog *Select QXP Doc in Export Hierarchy* will appear. Click on a document in the folder you want to XML check and select Open.

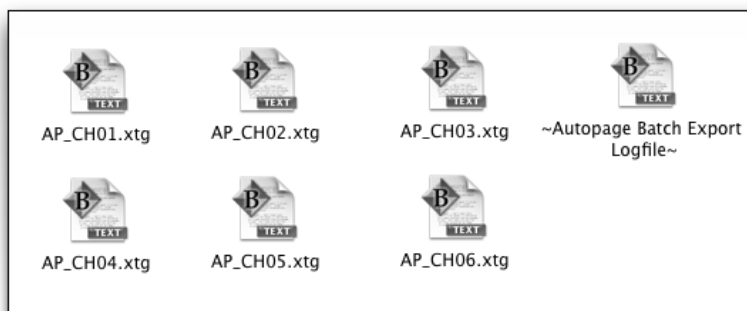


178 *Xtags Maximized*

- Each chapter will open up as the *Batch Export* utility runs. It will first run through the XML Export. If you do not own XMLxt, some of these export functions will not be available.



- Next, the *AutoTag Export Conversion Status* will happen. If there's a problem with the linking of an Autopage document, this will not complete to 100%.
- Once the export is complete, open the ~Autopage Export Folder~ within the folder you selected to run through the *Batch Export*. If everything ran as expected and you exported with Xtags, all of your chapters should contain the same name as your Quark documents followed by an .xtg (Xtags) extension. If a file doesn't export properly, check the ~Autopage Batch Export Logfile~ to identify the problem.



- Once the Xtags file (.xtg) generates, the files are now ready to run through a script to become XML files. Otherwise, they are similar to a marked-up file ready for import complete with all style information, overrides, etc.

This is where your knowledge of BBEdit's *Grep* and AppleScript comes into play. If you do not own a copy of BBEdit, I would suggest purchasing this. This is one of those programs I cannot live without. The price tag is very affordable and the power it gives to your text files is comparable to the power Xtags adds to QuarkXPress.

The text document will look like this when it exports. Remember, this document was not through any XML pre-tagged formatting.

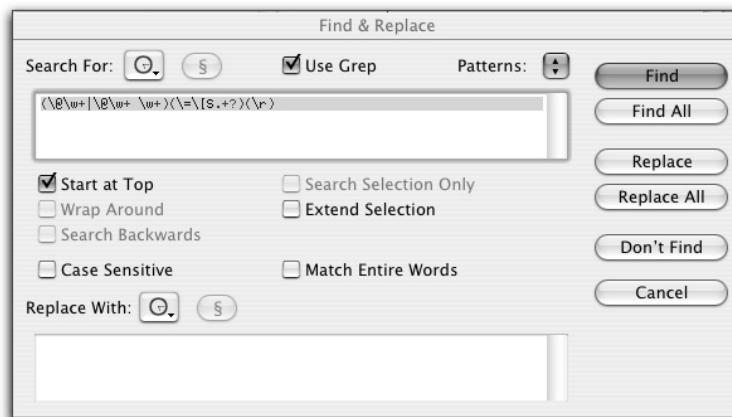
```
<v2.00><e0>
@NLN=<P><s100><t0><h100><z12><k0><b0><cK><f"Futura-Bold">
@H1=[S"","H1"]<*L><*h"H1"><*kn0><*kt0><*ra0><*rb0><*d0><*p(0,0,6,12,21,6,g,"U.S.
English")><P><s100><t0><h100><z16><k0><b0> <cK><f"Futura-Bold">
@NL1=[S"","NL1"]<*J><*h"Pre P1"><*kn0><*kt(2,2)><*ra0><*rb0><*d0><*p(18,-18,0,
13,9,0,g,"U.S. English")><*t(18,0,"1 ")><P><s100><t0><h100><z11><k0><b0>
<cK><f"Berkeley-Medium">
@CFT=[S"","CFT"]<*C><*h"Standard"><*kn0><*kt0><*ra0><*rb0><*d0><*p(0,0,0,30,0,0,g,
"U.S. English")><K><s100><t0><h100><z30><k0><b0><cK><f"Slimbach-Black">
@H1:
@NL1:
@CFT:
@CFN:Chapter 1
@CFT:Building Picture and Text Boxes
@TX1:When I was first introduced to Xtags almost 10 years ago, the picture and text box tags
were the big sell to me. I had no idea how much time I had wasted prior to realizing the
power that lies in these tags alone. <t1>A lot of automation awakenings happened for me
upon learning about Xtags.<t0>
@H1:Xtags Fields
@TX1:Xtags uses comma-separated fields to process the detailed information for importing
boxes and images. The following shows a complete string for importing art:
@CTR:<b1>[<{\<>ART FILE="56 GB Disk:Xtags Book:Images:Chapter 03:fg03_02.eps"
W="182.882pt" H="51.539pt" XS="100%" YS="100%"/<\>>}]<b0>
@H2:Xtags Issues
@TX1:When using Xtags to bring in photos or boxes, one of the biggest mistakes is ending
up with unnecessary information. When using Xtags, there are two ways to get the Xtags
strings. One is to type the fields in manually, or by grouping the boxes together, going to
<f"Berkeley-Italic">Edit:Copy Xtags Text<f"Berkeley-Medium"> and <f"Berkeley-
Italic">Paste<f"Berkeley-Medium">.
<t0>@NL:<*p(36,-36,0,13,5,0,g,"U.S. English")><t3> 4. <f"Berkeley-
Italic">height<f"Berkeley-Medium"> <t0>-<t3> This is the depth (height) of the text box. By
typing in the maximum depth of the art can be and adding a question mark (?), the box will
<@KT>shrink-to-fit<@ $p> to the depth of the art.[ch_end]
```

As you can see in the highlighted text, this is where Xtags throws in the fonts, picture boxes, paragraph overrides, tracking, and baseline shift information. This is only a partial example of what will show up. We will tackle this small example to show how we can make this content XML tagged instead of Quark ready content. There are a series of steps I go through. I will have the replacements to it's easier to follow along. It's best to record these into an AppleScript.

1. Remove all of the paragraph style sheet definitions at the start of the document. They look like this:

```
@CFT=[S"","CFT"]<*C><*h"Standard"><*kn0><*kt0><*ra0><*rb0><*d0><*p(0,0,0,30,0,0,g,
"U.S. English")><K><s100><t0><h100><z30><k0><b0><cK><f"Slimbach-Black">
```

This can be achieved by doing a *Grep* “Find and Replace”, which was discussed in Chapter 6. The `\w+` is any word character (a-z, A-Z, _, 0-9). This should cover them all unless the designer put in non-word characters into the style sheets. It’s a good practice to have “_” instead of spaces as well, such as “T_FIRST”. Here is the actual BBEdit “Find and Replace” window with performing a *Grep* search.



Due to the fact that I don’t have space to show this window each time, I’m going to just put in a “Search for” and a “Replace With”. The settings for the following examples will always be “Use Grep” and “Start at Top” unless stated otherwise. Here’s how this will be shown:

Search for:

`(\@ \w+|\@ \w+ \w+)(\=[S.+?])(\r)`

Replaced with:

Leave Blank

2. Delete all character style definitions at the beginning of the chapter. They appear like:

`@NLN=<P><s100><t0><h100><z12><k0><b0><cK><f"Futura-Bold">`

This is very similar to the paragraph style. The difference is that we are not matching against the “[S]” code, but rather the “<P>”, “<K>”, etc that starts off the character style definition. Here is the replacement:

Search for:

`(\@ \w+|\@ \w+ \w+)(\=[<D+>.+?])(\r)`

Replaced with:

Leave Blank

3. Delete the list of the style sheets that appear following the paragraph and character style sheets:

@H1:
@NL1:

You must be careful with this one and be very specific to ensure that you do not delete the style names throughout the text file. It is important not to put in a “.+?” here because it could go through the entire line if a “:” ended the line of text. Here’s the replacement:

Search for:

`(\\@\\w+|\\@\\w+ \\w+)(:\\r)`

Replaced with:

Leave Blank

4. Go through the document and decide what content is unnecessary to your objective. I typically go through the file and delete all tags that are not pertinent to the file. This includes the tracking, baseline shifts, paragraph overrides, etc. Here is the replacement to delete a series of these all at once:

Search for:

`(<t\\d+>|<-t\\d+>|<t\\d+\\.\\d>|<b\\d+>|<b-\\d+>|<k\\d+>|<-k\\d+>|<s\\d+>|<h\\d+>)`

Replaced with:

Leave Blank

This takes care of all of the tracking, kerning, shading, and horizontal scaling that would be overrides in the text file. The tracking, kerning, and baseline shifts have a negative and positive search.

5. There are other things throughout the file that need to be changed. The search above only wipes out those overrides. For example, any occurrence of paragraph overrides will need to be deleted like this:

`<*p(36,-36,0,13,5,0,g,"U.S. English")>`

This would need the following deletion:

Search for:

`(<*p\\(\\d+\\.+?English\\\"\\)>)`

Replaced with:

Leave Blank

I think you get the idea on the deletions that need to occur.

182 *Xtags Maximized*

6. Next, I start mapping the XML to the necessary DTD based on the style sheet names. The first thing I will need in the XML file is the starting tag. This is very easy with *Grep*. The “\A” code matches at the beginning of the first line. We know that this needs to be the tag for the XML. Depending on your DTD, different possibilities can occur. We’ll just put in the a standard tag for demonstration purposes:

Search for:

\A

Replaced with:

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

End Result:

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

7. The first tag following the XML tag will be the chapter number style “CFN” beginning style sheet. The text file shows:

@CFN:Chapter 1

To get this to work with a *Grep* “Find and Replace”:

Search for:

(\\@CFN:)(Chapter)(1)(\\r)

Replaced with:

<chapter num\\=\"3\" id\\=\"ch.00.0\\3\\\">\\4

End Result:

<chapter num\\=\"3\" id\\=\"ch.00.0\\3\\\">\\4

8. The title is a lot easier because you are only dealing with the start and end of this. There is no extra handling needed.

Search for:

(\\@CFT:)(.+?)(\\r)

Replaced with:

<chapter title>\\2<\\chapter title>\\3

End Result

<chapter title>Building Picture and Text Boxes</chapter title>

You can get a lot of mileage on the above search for `(\\@style)(.+?)(\\r)`. This is basically just looking at the style sheet, any content within, and the return. Most styles will read off of this, especially paragraphs. I've had books where 60% of the content was handled by matching the style sheet names and using this code.

More Intricate Coding

Going beyond the basics, we will use the Xtags exported tags to change lists, boxes, style enhancements, and other tags. I think it is best to make your own BBEdit file of the content I have and try each of these examples and practice along as I show each of these examples. This should give you a better understanding of the results you will need to produce.

Let's start with some inline elements. This could be italicized words, key terms in text, among many other possibilities. Let's first look at the example:

@NL: 1. **<f"Berkeley-Italic">height<f"Berkeley-Medium">** –
This is the depth (height) of the text box. By typing in the maximum depth of the art can be and adding a question mark (?), the box will
<@KT>shrink-to-fit<@\$p>.

We have three different tags that will be altered. We need a list, an italic, and a key term. We can do all of these by following the *Grep* replacements. Let's start with the key term.

Search for:

`(<\\@KT>)(.+?)(<\\@$p>)`

Replaced with:

`<term>\\2</term>`

End Result:

`<term>automation</term>`

You do not want to just delete font information without looking closer at your content. This is why it is important to really study your document before commencing. This could be a very costly mistake if you fail to do this. For starters, you'll want any `<italic>` and `<bold>` content to have the proper codes surrounding the words. You do not want to just go through and delete these, so it's very important to look through the *Font Usage* to see what fonts are being used throughout the document.

Be aware that the pager can sometimes be careless. I find it best to do the font substitutions after all the general deletions are made, but prior to doing the style sheet information. In the example below, I make sure I am only covering the fonts that will close off the italics, bold, etc. such as "Times", "Optima", and

184 *Xtags Maximized*

“Helvetica”. I don’t want it to pick up on “Symbol” or “Mathematical Pi”. That is why I handle them like this:

Search for:

```
(<f\"Berkeley-Italic\">)(.+)(<f\"Berk.+? | Helv.+? | Opti.+?\">)
```

Replaced with:

```
<italic>\2</italic>
```

The numbered list can be handled a number of different ways depending on the DTD. For this example, we will handle it like this:

Search for:

```
(\@NL1:)(\t)(1)(\t)(.+?)(\r)
```

Replaced with:

```
<numberList>\6<listItem>\5</listItem>\6
```

End result of all three examples:

```
<numberList>
```

```
<listItem><italic>height</italic> – This is the depth (height) of the  
text box. By typing in the maximum depth of the art can be and  
adding a question mark (?), the box will <keyterm>shrink-to-  
fit</keyterm> to the depth of the art.</listItem>
```

Each additional list item would just be <listItem>text</listItem>. It is very important that the pager uses a style like “NL2” for the closing line of the numbered list, or errors will occur during the XML validation because it cannot locate the ending numbered list code. If the document was handled correctly to start with, the search and replacement would be:

Search for:

```
(\@NL2:)(\t)(\d+)(\t)(.+?)(\r)
```

Replaced with:

```
<listItem>\5</listItem>\6  
</numberList>
```

Keep in mind that whatever tag starts in XML, must always close. If you have a starting <numberlist>, you need to close it with a </numberlist>. The same works with HTML and actually makes the entire process fairly easy to keep track of.

You could also be tricky in your searches and use the power of *Grep* to find any list items that were incorrectly handled throughout the process. I find this to be a good method of error checking. There are many times when

the pager will have a numbered list start off with a “nl1”, then the remaining are “nl”, but when they get to the last one they do not put a “nl2” on it because a head follows and they don’t need the adjustable space. This is where it is very difficult to work with the XML because it is difficult to determine where the numbered list ends.

Here’s an example of how the numbered list may look upon export.

```
@nl1: 1. This is the first entry
@nl: 2. This is the second entry
@nl: 3. This is the last entry
@h1:This is the new H1 head.
```

The problem here is that the numbered list in the middle will end up with a closing `</listItem>` on it but without the closing `</numberList>` on it. There is a way around this by performing a *Grep* replacement prior to doing the actual replacement for the rest of the list.

Search for:

```
(\\@nl:)(.+(?)(\\r)(\\@)([A-Z] | \\d+)([\\^L] | \\d+ | _)(.+(?)(\\r)
```

Replaced with:

```
<listItem>\\2<\\listItem>\\3<\\numberList>\\3\\4\\5\\6\\7
```

End Result:

```
<listItem> 3. This is the last entry</listItem>
</numberList>
```

This is going through and finding an “nl” style, but making sure that what ever follows is anything but an “nl” style. The highlighted area shows the first letter of the style can be any letter or number, and the `[^L]` matches on any character except an “L”. This keeps it from matching on “NL”, but can match on any other style combination.

The number remains after doing this replacement, but it can be removed from the `<listItem>` as easy as doing this replacement:

Search for:

```
(<listItem>)(\\t\\d+\\.\\t)(.+(?)(<\\listItem>)
```

Replaced with:

```
\\1\\3\\4
```

You will want to make these replacements before you change the rest of the “`<listItem>`” styles. It is easier to match upon the Quark style because the `<ListItem>` tag can be part of the unnumbered list, lettered list, etc. To save a validation time because of closing tag errors, it is a good workflow idea to write *Grep* replacements for each list tag to ensure there is a closing tag.

Working with Heads and Run-in Heads

Heads such as h1, h2, h3, etc., are all fairly easy to deal with, but you may run into mathematic books that have numbered heads that require additional coding. It all depends on the content. A H1 with no extra coding would appear in the XML file as:

```
<h1>This is the head</h1>
```

However, a numbered H1 that has a “3.4” for the number could need to take on this type of a replacement:

Search for:

```
(@H1:)(\d+\.\d+)(\t)(.+)(\r)
```

Replaced with:

```
<h1 id=\"2\">\4</h1>\5
```

End result:

```
<h1 id="3.4">New Directions in Automation</h1>
```

A run-in head, such as an “h3” head that has a paragraph direction following would need to take on this type of treatment:

Search for:

```
(@H3:)(<\@h3>)(.+)(<\@$p><\f><\f>)(.+)(\r)
```

Replaced with:

```
<h3>\3</h3>\r<para>\6</para>\7
```

End result:

```
<h3>Roundtripping</h3>
<para>The conversion of data into a different configuration and
back to original structure without loss of content.</para>
```

Images

This is where Xtags really comes into action with the export. It will export the entire path to the art which makes the repurposing that much easier. Based on the information below, I can tell this piece of art is a numbered figure and it is the second image.

```
[{<\>ART FILE="56 GB Disk:Xtags Book:Images:Chapter
03:fg03_02.eps" W="182.882pt" H="51.539pt" XS="100%"
YS="100%"/<\>}]
```

As a result, my search and replacement is that much easier:

Search for:

```
(\[{<\<>ART FILE=\\"56 GB.+?)(Chapter
03:)(fg)(\d+)(\d+)(.eps)(\" W\\=\"182.882pt\" H\\=\"51.539pt\"
XS\\=\"\+.+?\%\" YS\\=\"\+.+?\%\"\\)(<\>}})
```

Replaced with:

```
<figure num=\"6\" id=\"4\\.6\"><ART
FILE\\=\"\3\4\5\6\7\8></figure>
```

End result:

```
<figure num=\"02\" id=\"03.02\"><ART FILE=\"fg03_02.eps\"
W=\"182.882pt\" H=\"51.539pt\" XS=\"100%\" YS=\"100%\"/></figure>
```

Typically images will match on the style name such as “@fqn”, but for this example, I wanted to show how the figure could be extracted just out of this information. To do it the other way would start by simply following the (@style:)(.+?)(\r) type of replacement and then just follow similar steps to what is shown above.

Searching Through the File

One of the last steps I like to make is going through the file and looking for content that doesn’t need to be in the file. For example, in Quark, if the pager does a soft return, a “<\n> will appear in the file. You do not just want to delete this because the word following may end up against the other word causing a typographical error such as “walkinto”. Instead I do the following:

Search for:

```
<\n>
```

Replaced with:

```
two literal spaces
```

Putting in two literal spaces is easy to go through afterwards and do a search for the two spaces and then replace with 1 space.

Other changes I make is looking for extra returns. This can be accomplished by simply searching for “\r\r+” and then replacing with “\r”. I like to have as clean of a file as possible.

I will also look for spaces or tabs before the line ending that are unnecessary. This can be accomplished by simply searching for “(\s+)(\r)” and replacing with “\r”.

Closing the File

In an XML or HTML workflow, there always needs to be a start and end code. BBEdit can put in the beginning with the “\A” code and the closing by using the “\Z” code. Here’s an example:

Search for:

\Z

Replaced with:

\r<\chapter>

Based on the replacements I displayed here, our original file shown at the start of the chapter would now reflect the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
<chapter num="1" id="ch.00.01">
<chapter title>Building Picture and Text Boxes</chapter title>
<para>When I was first introduced to Xtags almost 10 years ago, the picture and text box
tags were the big sell to me. I had no idea how much time I had wasted prior to realizing the
power that lies in these tags alone. A lot of automation awakenings happened for me upon
learning about Xtags.</para>
<h1>Xtags Fields</h1>
<para>Xtags uses comma-separated fields to process the detailed information for importing
boxes and images. The following shows a complete string for importing art:</para>
<figure num="02" id="03.02"><ART FILE="fg03_02.eps" W="182.882pt" H="51.539pt"
XS="100%" YS="100%"/></figure>
<h2>Xtags Issues</h2>
<para>When using Xtags to bring in photos or boxes, one of the biggest mistakes is ending
up with unnecessary information. When using Xtags, there are two ways to get the Xtags
strings. One is to type the fields in manually, or by grouping the boxes together, going to
<italic>Edit:Copy Xtags Text</italic> and <italic>Paste</italic>.</para>
<numberList>
<listItem><italic>height</italic> – This is the depth (height) of the text box. By typing in the
maximum depth of the art can be and adding a question mark (?), the box will <term>shrink-
to-fit</term> to the depth of the art.</listItem>
</numberList>
</chapter>
```

We’ll cover more on markers in floating elements in this next section.

Know the Content

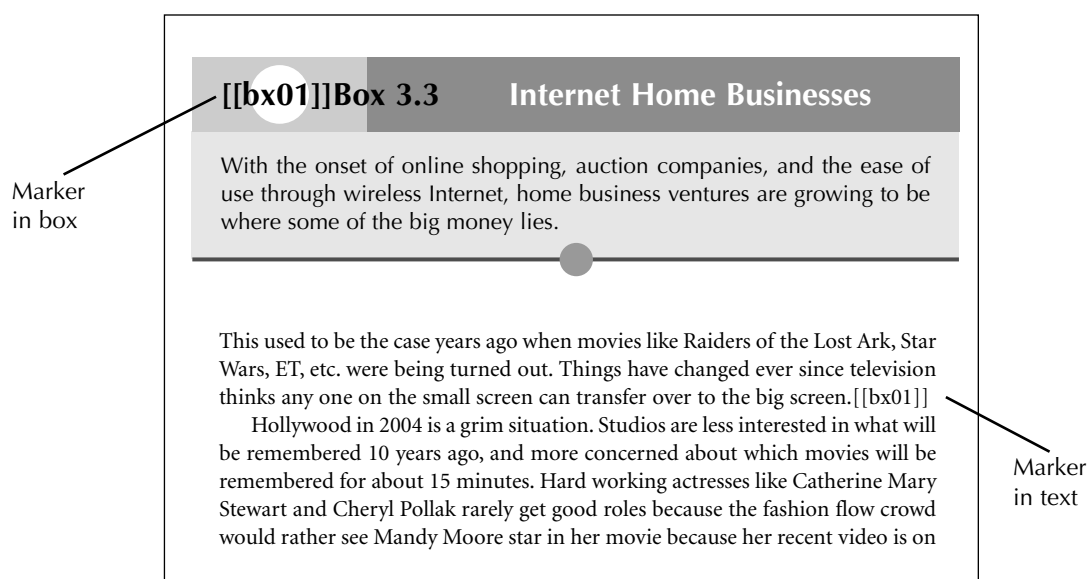
Doing repurposing will be a struggle if you do not know what the end result should be. If you want to make an HTML document out of the material, it is important to have a working file of one finished chapter so you will know

what style or content needs to be mapped to which code. That is the best way to learn what needs to become what.

I actually had to take XML coded tables and turn them into HTML tables. At first I thought this was going to be a struggle, but once I saw what the end result was, it only took me a short amount of time to figure out an approach. Once I set it up, the majority of tables were done in minutes with very little handwork. This was from knowing the content ahead of time.

Floating Element Markers

Earlier in this chapter I said that any floating elements need to have a marker. The reason for this is that if a box is placed in text, but it doesn't have something referring to it, then it would be very difficult during the repurposing to figure out where it actually goes without referring to the original paged file. By the text and the box having a marker, a script can be written that can move the type in the text to the area where referenced. Here's an example of how this might look in the file:



You can see the box is floating at the top of the page and below is the text. Since the entire text chain exports first and then the boxes and other floating elements, the box will export far from its targeted positioning. That is where the two markers will bring this together. For boxes, you should use a consecutive box marker such as `[[bx01]]`, `[[bx02]]`, etc., to place in the box and a matching marker near the reference area during post production on a duplicated file. All of this is possible to do because of the content Xtags provides during the export.

190 *Xtags Maximized*

The first step is for the `[[bx#]]` to match in the floating element and create up the starting box tag. Here's an example of how this would be done.

Search for:

```
(@BX9T:)(\[ [bx)(03)(\])(< \@bxt>Box 3.3< \@ $p>\t)(.+?)(\r)
```

Replaced with:

```
<box number=\="3"><title>\6</title>\7
```

End Result:

```
<box number="03"><title>Internet Home Businesses</title>
```

This is where I rely on the magic of BBEdit combined with AppleScript. The AppleScript will go through the text file and find the entire box and then cut it and paste it in the place of the referenced `[[bx#]]` placeholder. It is matching on the `<box number="03">` tag and the content following:

```
tell application "BBEdit"
    activate
    try
        find "(<box
            number\\=\\"03)(\\\".+?<\/title>\\r<para>.+?<\/box>|\\\".
            +?<\/title>\\r<para>.+?\\r<para>.+?<\/box>)" searching in
            text 1 of text document 1 options {search mode:grep, starting at
            top:true, wrap around:false, backwards:false, case sensitive:false,
            match words:false, extend selection:false} with selecting match
        cut selection
        find "\\ [bx03]" searching in text 1 of text document 1 options {search
            mode:grep, starting at top:true, wrap around:false,
            backwards:false, case sensitive:false, match words:false, extend
            selection:false} with selecting match
        paste
    end try
    replace "(<\/para>)(<box)" using "\\1\\r\\2" searching in text 1 of text
        document 1 options {search mode:grep, starting at top:true, wrap
        around:false, backwards:false, case sensitive:false, match
        words:false, extend selection:false}
    save text document 1
end tell
```

You would have to repeat the AppleScript step for each box, and the script actually gets really long, but once it is written, it can be reused for other books. As long as “try” and “end try” are included, if it does not match the number of boxes, the script will continue to the next step and not generate an error. One thing we didn’t go over is how to get the closing `</box>` tag. This is through the power of Xtags. As long as you have the entire box grouped, it will end with a grouping tag. In this case it was “`<&g(1,2,3,4,5,6)>`”. Knowing this, all I need to do is substitute this out for the closing `</box>` tag. This

also picks up a picture box string and a line tag string, but with specific information inside them. For example, the picture box has a (,o) for an oval, as well as a “K,40, shading, so this will be easier to locate the content.

```
<para>With the onset ... big money lies.<&te><&lbu(0,75,0,
287,or,,1.5,,70,,,,,><&pbu2(138,69,12.5,12.5,,,,,(o),(n),(100),
,K,40,(,,),m,,,,,,,"",,,)><&g(1,2,3,4,5,6)></para>
```

Knowing this, we can break this code down quite easily to delete all of the Xtags information no longer needed, but still matching on it so we can make this change:

Search for:

```
(<para>.+?)(<&te>)(.+?\(,o\).+?K,40.+?)(<\&g\1,2,3,4,5,6\>)
(</para>)(\r)
```

Replaced with:

```
\1\5</box>\6
```

End Result:

```
<para>With the onset ... big money lies.</para></box>
```

Here is an example of how the box would position upon export in the text file. If you notice, there are a few paragraphs and then the box appears. Typically, the entire text chain would be first and the floating elements would all gather at the end of the file. For demonstration purposes, we made this easier. In the finished file, the box needs to position right below the first paragraph where the marker is.

```
<para>This used to be the case years ago when ... the big
screen.</para>[[bx01]]
<para>This is another paragraph of text.</para>
<box number="03"><title>Internet Home Businesses</title>
<para>With the onset of online shopping....</para></box>
```

After the AppleScript is run against the text file, it will pull the entire box from its current position in the text and move it to the line following the paragraph. Here is the result:

```
<para>This used to be the case years ago when ... the big
screen.</para>
<box number="03"><title>Internet Home Businesses</title>
<para>With the onset of online shopping...</para></box>
<para>This is another paragraph of text.</para>
```

This treatment will work for tables, figures, side elements, or any floating objects that you need a marker for. I realize it takes time to go through and mark these initially, but it is much easier when running the scripts to just let them do their magic instead of cutting and pasting by hand. This is where the time savings will come in.

Hyperlinks

Quark 6.5 and higher makes great use of hyperlinks. If used correctly in the paging file, this will save time later. I would insist on the URLs being either treated with a character style such as <@URL>web address<@\$p> or using the *style:hyperlink* option. If using a character style, the process is similar to what has been shown in this chapter. If using a hyperlink, Xtags will export this as:

```
<A(3,"HYPB",\#002\#000\#000\#000)[2]>www.highvolumeexpress.com
<A(3,"HYP\#034",\#000\#001)[1]>
```

This can be repurposed by writing a script that finds the beginning and the end of the hyperlink and put a <URL> around the type.

Search for:

```
(<A\ (3, \"HYPB.+?\>)(.+?)(<A\ (3, \"HYP.+?\>)
```

Replaced with for XML:

```
<URL>\2<\URL>
```

Replaced with for HTML

```
<a href=\"http:\\\2\"><\a>
```

The process I've shown throughout this chapter has more benefits if you are doing a series and you can repeat these steps throughout many books. The setup time pays off over the series. Many of the “Search” and “Replacements” shown here can be used for many projects. I repeat the replacements for tracking, baseline shifts, etc. in each book. It's a great practice to build a standard AppleScript containing what you will be using from one project to the next.

If you are a publisher and your style sheets are always the same, using a FileMaker Pro database to store all of the style sheet replacements is a great option. FileMaker Pro has the ability to call AppleScripts from BBEdit or other programs allowing more functionality. You can be more specific with pulldown menus. We will cover this more in the next chapter.

While I only displayed a small portion of this, it should give you a good idea of what is possible and how the exported Xtags data can give you the opportunity to bridge the gap between the Quark file and an XML, HTML, or other coding possibilities. Once you learn *Grep* and have an export of the Xtags file, nearly any end result is within your reach.

The chapter title graphic features a large, stylized number '9' on the left, with the word 'CHAPTER' in a smaller, bold, sans-serif font directly beneath it. To the right of the '9', the title 'Xtags and FileMaker® Pro' is displayed in a large, bold, sans-serif font. The entire graphic is set against a dark, textured background with a grid of light gray lines and a subtle glow effect around the text.

9 Xtags and FileMaker® Pro

CHAPTER

If you are not using FileMaker® Pro you are really missing out on a world of new possibilities. I've found that FileMaker Pro has made automation even easier and has opened up new ways to take scripted material and use it in a user-friendly way. I've asked many people if they use FileMaker Pro and they use it for tasks like tracking their time, inventory, and for listing contact information. Meanwhile this person is setting up all of their Xtags strings by hand and having a coding department type in the figure names. This can all be tackled using this program.

I've also seen people who will at least use BBEdit and perform the "Grep" searches to get their material in, but they are still doing more manual manipulation than they should be. When you can use FileMaker Pro to do the same thing and incorporate all of your "Grep" searches, AppleScripts, but make them modifiable based on field information, the door opens wide and faster workflows will develop.

I have spent much time working on databases geared for Xtags, Autopage, standard designs, XML tagging, and other publishing areas that were normally handled manually or not as efficiently as they could be. For example, I knew that glossary terms were difficult for one publisher in particular. They would send all of the final copy and pay someone to look through each page, circle the glossary term and definition, and then have someone code it for the end matter...this is not cost effective. Instead, I made a workflow where the following happens:

1. Using the Batch Export feature in Autopage, the entire text is exported to an xtags (.xtg) file.
2. Through FileMaker Pro, I go through and remove the unnecessary content, then pull all of the terms that match the style names for that book.
3. These terms are put into a separate file where they are reformatted with Xtags to fit the end matter file.

4. Through an AppleScript, all of the terms are alphabetized and the duplicate terms that may have occurred in more than one chapter are removed.
5. The new file is then ready for import into QuarkXPress.

Try doing that by hand and see how long it would take. The entire process of running this through FileMaker Pro will take 5 minutes or less after the initial Autopage *Batch Export* if set up correctly.

■ Importance of Databases

When I get sold on a program, it's hard to steer me away from it. I have software that I primarily work with including Xtags, Autopage, AppleScript, BBEdit, and FileMaker Pro. I have two databases, in particular, that I want to show you. I'm not going to break down every single detail because that would be better served in another book, which I might just do soon. But, if you have the software listed above, you will find yourself with a new array of areas to begin working on.

One of the reasons I find FileMaker Pro to be so important is that it can store thousands of replacements, make enhancements to fields and other content using intricate scripts, and the user-friendly interface. For example, if you were working with XML content, you could want to make all of the boxes be a variable like:

```
<box num="1" ID="box1_0201">
```

The 0201 would be broken down as “02” (Chapter No.) and “01” (Box No.) If this was a custom job, how could you tell the entire document to change the chapter number to “12” instead of “02”? Through BBEdit, you could do a series of searches for the figures, marginal terms, boxes, tables, etc. and change them that way. Through FileMaker Pro, you could just change the pull down menu for the *Chapter* field to “12” and then run a script to do this against the entire XML file. This is something that we will look at as this chapter progresses.

Level of Understanding

If you have no understanding of FileMaker Pro, you can still read this chapter, but I think it would be a good idea to familiarize yourself with the program and some of the features. I'll be jumping around here quite a bit, so if you want to know the program, I'd suggest either buying a book on it, doing an online course like on www.vtc.com, or just working in it.

What I want to show, more than anything, is how we can take Xtags content and make it user-friendly where you really don't have to count the commas or know all of the coding options in any tag. The database will just do this for you and you will get the end result in a fraction of the time. Building the database takes time, but it is a rewarding experience when it is finished.

I am going to show two databases in this chapter. We'll start with the *Xtags Creator* that I created that builds picture and text boxes for figures, unnumbered figures, and photos and adds the references and Xtags strings to the text file. By breaking down the logic behind this, you will quickly see where you can take FileMaker to improve your current workflow.

Database 1 — Xtags Creator

XTAGS Creator

Art Placement Option 1 ☒

Job Number Author Last Name ISBN

Path to Art

Art Name Type of Art ☒ Figures ☐ Photos ☐ Unnumbered

of Grouped Boxes Autopage Usage ☒

Side Caption ☐ **Bottom Caption** ☒ **Top Caption** ☐

Figure 1.1
A view of a skyscraper.



Figure 1.2
A view of New York, pre 911.



Figure 1.3
A skyline city view.



Image Width and Depth

Art Maximum Width ☒ Shrink to fit

Art Maximum Depth ☒

Caption Width and Depth

Caption Width ☒ Relative

Caption Depth

Space from Art to Caption

Xtags Results

```
[[[fg1]]<&pbu2(300 B,0,24p?,53p?,,,,n,(.5),(n),
(100,100),,"Black",0,,m,100,100,0,0,0,0,"Macintosh HD:Users:
Automation Folder:Filemaker Files_NEW:Xtags Images:
[[fg2]]",,,,)><&tbu2(-300 TL1,0,24p,(4p,R,0,1'),,,,n,(n),
(100,100),,N,0,,,,,>
[[fg3]]<&tc><&g(1,2)>
```

Images: Photos.com

Option 2 Option 3 Option 4

Text Importing Opt 1 Advanced Run Script

Shown here is how the opening page of the *Xtags Creator* database will appear. This is a database I created, but you can build a similar interface by creating the art through Adobe Photoshop or Adobe Illustrator. There is a lot happening in this program. I'm going to give a little background about FileMaker Pro before going into the *Xtags Creator*. Each button or place to fill in information is known as a *field*. You need to define each field with a unique name to be used throughout. This is how FileMaker Pro interprets the data.

Each record is like its own job. You can store a job number as a record for each author you work with. There are scripts that can take the field information and manipulate it to do nearly anything you need. There are also buttons, like in the lower right of the program that can run scripts, take you to other pages, load images, or however else you script these.

Starting at the top of the page, the first thing to notice is we're on the "Art Placement Option 1" page. With this checkbox clicked to the right, it activates everything for the type of art selected. I have this scripted where the program will read through all of the fields and, based on that information, decides how the Xtags strings should be formatted. At the end, I will run a script that will create the Xtags Translation Table based on the information that the user specifies in the fields.

A series of fields are included at the top of the page including Job Number, Author Last Name, ISBN, Path to Art, Art Name, Type of Art, No. of Grouped Boxes, and Autopage Usage. These are critical to the success of the job. For example, we have several different options for the "Art Name" field. There is a value list that you can define under *File:Define:Database* that stores which options can be in a pull-down menu. Based on the possibilities for art, I realize that four options could occur. The possible art names will be:

```
f0101.eps
fg_0101.eps
author_f0101.eps
isbn_f0101.eps
```

Based on this information, within the "Art Name" *field*, I select the "f0101.eps" pull-down selection. A script will look at this definition, and it will add expanded tag information. The markup department will need to put a small variable before and after the figure so the database can replace the information correctly. It is important that these are numbered sequentially for each image. For example, Figure 5.1 will be `[[f01x]]`, Figure 5.2 will be `[[f02x]]`, etc. Here is an example of how the tags should be coded:

Callouts

Figures	<code>[[f01z]]</code>
Photos	<code>[[p01z]]</code>
Unnumbered	<code>[[u01z]]</code>

Start of Caption

Figures	<code>[[f01x]]</code>
Photos	<code>[[p01x]]</code>
Unnumbered	<code>[[u01x]]</code>

End of Caption

Figures	<code>[[f01y]]</code>
Photos	<code>[[p01y]]</code>
Unnumbered	<code>[[u01y]]</code>

These markers will be used to add data based on the information scripted inside the program. It is much easier for the coding department to add these

small variables rather than actual art names. Here is an example of how the text will be coded using these markers:

```
[[xtags]]
@tx1:This is text. Unnumbered Callout [[u01z]], Photo Callout [[p01z]],
and Figure Callout [[f01z]]

@fgn:[[f01x]]Figure 3.1 This is the caption[[f01y]]
@fgn:[[f02x]]Figure 3.2 This is the caption[[f02y]]

@fgn:[[p01x]]This is the first photo caption[[p01y]]
@fgn:[[p02x]]This is the second photo caption[[p02y]]

@fgn:[[u01x]]This is the first unnumbered caption[[u01y]]
@fgn:[[u02x]]This is the second unnumbered caption[[u02y]]
```

This text file will be imported into the database. An AppleScript within the database is run to match against these codes. The file exports out of the database into a new BBEdit file and makes all of the replacements. Here is what the file will then look like.

```
<&tt2"Williams.ttl">
@tx1:This is text. Unnumbered Callout [[AR U01 V=2 I=Y]], Photo
Callout [[AR P01 V=2]], and Figure Callout [[AR 01 V=2]]
@fgn:[[fg1]]Chapter 04:fg_0401.eps[[fg2]][[A 01]]Figure 3.1 This is the
caption<&te><&g(1,2)>
@fgn:[[fg1]]Chapter 04:fg_0402.eps[[fg2]][[A 02]]Figure 3.2 This is the
caption<&te><&g(1,2)>
@fgn:[[ph1]]Chapter 04:ph_0401.eps[[ph2]][[A P01]]This is the first
photo caption<&te><&g(1,2)>
@fgn:[[ph1]]Chapter 04:ph_0402.eps[[ph2]][[A P02]]This is the second
photo caption<&te><&g(1,2)>
@fgn:[[un1]]Chapter 04:ua_0401.eps[[un2]][[A U01]]This is the first
unnumbered caption<&te><&g(1,2)>
@fgn:[[un1]]Chapter 04:ua_0402.eps[[un2]][[A U02]]This is the second
unnumbered caption<&te><&g(1,2)>
```

So the question you probably have is “how does this happen?” I will break it down one step at a time.

Scripting Option

One thing to understand about FileMaker Pro is that it has a very powerful scripting engine called *ScriptMaker* that gives you the ability to alter the *field* information in the program to your specifications. For this database, I elected to use AppleScripts within *ScriptMaker* to go into BBEdit and use “Grep” to make these replacements to the text file.

There are two options for AppleScripts. One can be copied directly out of AppleScript and in the *Perform AppleScript* window it can be pasted into the *Native AppleScript* option. Or if you want to use the data in the fields, then you would use the *Calculated AppleScript* option where you can grab information out of the fields and really do some intricate replacements to the text. I tend to do the majority of my work out of the *Calculated AppleScript* option because I can achieve replacements that would normally be handled manually.

The Xtags Translation Table Code

This is one of the most simple of the replacements, but it is a very important one. The markup department will code in `[[Xtags]]` at the top of the file. A script will need to be written to change this. It will read off the field that says “Author Last Name”. It is important to type the name correctly in here because a lot of script steps will read off of this field:



During the replacements, the information in the *Calculated AppleScript* will interpret this information:

```
"tell application "BBEdit"¶" &
"activate"¶" &
"replace "\\[[Xtags]]\\" using "<\\&tt2\\" & Xtags
Creator::Author Last Name & "\\\\.ttl\\">" searching in text 1 of text
document 1 options {search mode:grep, starting at top:true, wrap
around:false, backwards:false, case sensitive:false, match words:false,
extend selection:false}¶" &
"end tell"
```

This is very similar to the way a normal AppleScript would be written with the addition of an extra “\” in front of delimited characters. What I’m telling AppleScript to do is to take “`[[Xtags]]`” and replace it for the “`<&tt2"Author Last Name.ttl">`” which in this example will be “`<&tt2"Williams.ttl">`”. This is a basic replacement, but you can already see where the field names can enhance your AppleScripts.

Setting up the Database

Setting up the database correctly is important for the end user. That is why whoever creates the database really needs to know their Xtags well, so they can capture all of the different options for each field. This first page is showing options such as the image width and depth, caption width and depth, etc., but the *Advanced Options* has all of the other fields. We’ll touch on this in a few pages.

I will say that, for the most part, the information on the first page is what you will be changing most often. Generally, the advanced options do not change that often unless you have a real need for them.

You will want to customize a program like the *Xtags Creator* to best fit the needs of your company. I do feel what I'm showing here is a good working model because it works well for me and there isn't much I would want to add to it at this point. If you are working with several different publishing companies, it can be a major time saver just in figure naming.

Figure Names

Figure naming is where a lot of power can come into play without having to develop an additional art log. If you do composition for other companies, chances are you will have many different options needed for art naming. One company may use something like "fg0102.tif" while another may have "55225_fg0102.tif" which would be the last 5 of the ISBN and then the figure number. They may use this later for archiving purposes or repurposing, so the naming is critical. As a result, you will want to have the *Calculated AppleScript* be able to swap this out based on which figure you need. That is why the field "Art Name" is important because the naming is specified here:



For this example, we'll say we are going to use the first 3 letters of the author's last name in the name of the art. In this example, our targeted result will be "wil_fg0101.eps".

To set this up, a calculated AppleScript would need to be written. I will demonstrate how the program will choose between two different options by using the "Case" option which is basically doing an "If" statement for as many options as needed. For this example, we'll only use two. One is to achieve the "fg0101.eps" naming and the other the "author_fg0101.eps". The highlighted areas show how this is reading off of the fields. The fields being used are "Chapter No", "Autopage Yes No", and "Art Name".

```
"tell application \"BBEdit\"¶" &
"activate¶" &
Case (Xtags Creator::F1 Art Name = "fg0101.eps";
  "replace \"(\\[\\[\\[f)(\\[\\[d+|\\[\\[d+[a-e])(X)]\" using
  \"\\[\\[\\[\\[fg1]]Chapter \" & Xtags Creator::Chapter No & \":f\" &
  Xtags Creator::Chapter No & \"\\[\\[2\\[\\[.eps\\[\\[\\[\\[fg2]]\" & If (Xtags
  Creator::F1 Autopage Yes No = \"On\" ; \"\\[\\[\\[\\[A \\[\\[2]]\" ; \"\" ) & \"\"
  searching in text 1 of text document 1 options {search mode:grep,
  starting at top:true, wrap around:false, backwards:false, case
  sensitive:false, match words:false, extend selection:false}¶";
```

```


Xtags Creator::F1 Art Name = "author_f0101.eps";
"replace \"([\\[\\f(\\d+|\\d+[a-e])(X))\" using
\\[\\[\\[fg1]]Chapter " & Xtags Creator::Chapter No & ":" & Left (
Lower (Xtags Creator::Author Last Name) ; 3 ) & "_f" & Xtags
Creator::Chapter No & "\\2\\[\\[fg2]]" & If (Xtags
Creator::F1 Autopage Yes No = "On" ; "\\[\\[A \\2]]" ; "" ) & "\"
searching in text 1 of text document 1 options {search mode:grep,
starting at top:true, wrap around:false, backwards:false, case
sensitive:false, match words:false, extend selection:false}¶";
)
& "end tell"

```

What this is doing is saying that if the Art Name = “f0101.eps” to go through and find the [[f01x]], [[f02x]], etc. and change it to:

```
[[fg1]]Chapter 03:f0301.eps[[fg2]]Caption Text
```

If Autopage is being used, then a button will be clicked as shown here:



If this button is selected, it will activate this “If Statement”:

```
If (Xtags Creator::F1 Autopage Yes No = "On" ; "\\[\\[A \\2]]" ; "" )
```

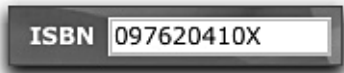
This creates the [[A 01]] Autopage tag marker based on the figure number.

The “author_f0101.eps” is much different because it is going to take the first 3 letters of the author’s last name and make it lowercase. This is executed by using this tag which says Left (text; 3) which is the first 3 left characters of the field, and using lowercase on that match:

```
Left ( Lower (Xtags Creator::Author Last Name) ; 3 )
```

This will produce this result “wil_f0301.eps”. It is very important that the art department names your art consistently for this to work. The “.eps” or “.tif” is not as much of a concern because we can run a script to determine this, but it is important that they do not add spaces, hyphens, etc. Anytime you are importing art, this is important, but if you provide markup with a list of names, it is easy to get the correct naming. We are letting the script generate the name, so it’s essential that it is accurate from the start.

I wanted to show one last example of naming by using the ISBN field that I created. Here is the ISBN field.



This field displays only 10 digits, where this will be changing to 13. You'll want to consider that when making this field. What we want to achieve here is to make a script step that will take the last 5 digits of the ISBN. This is very similar to the "Left" example above, but instead, we will be using "Right" and "5" characters to pick up the "0410X" for this art name.

Right (**Xtags Creator::ISBN**) ; 5)

Type of Art

Towards the top of the database, you will see a section that says "Type of Art" and the selections are *Figures*, *Photos*, and *Unnumbered Figures*. Depending on the positioning of the image and caption depends on which is selected:

Type of Art ☐ *Figures* ☒ *Photos* ☒ *Unnumbered*

If the design calls for each type of art to be handled the same, all three can be clicked. However, if you need a different type of treatment for the "Photos", you will need to go to "Art Placement Option 2" and create the Xtags for the photos. This is how the database keeps everything organized.

This is essential for the art to import correctly. Each art type will create the Xtags definitions to be used later in the translation table. Depending on which placement option you choose, will impact the tags that are generated. For example, if you use Art Placement Option 1, the database will generate:

[[fg1]] and [[fg2]] for the figures

If you select Art Placement Option 2, the database will generate:

[[fg4]] and [[fg5]] for the figures

The script will generate these tags for the translation table:

```
[[fg1]] <&pbu2(300 B,0,24p?,53p?,,,,n,(.5),(n),(100,100),,"Black"
,0,,m,100,100,0,0,0,0,"Macintosh HD:Users:Williams:
[[fg2]] ",,,)><&tbu2(-300 TL1,0,24p,(4p,R,0,1"),,,,n,,(n),
(100,100), ,N,0,,,,,,,,,>
```

This is why you'll need to create three different placement options in case all three types of art need different handling.

Art Placement Options

At the bottom of the page, you will see three buttons labeled as *Option 2*, *Option 3*, and *Option 4*. These are separate pages for new Xtags definitions.

Option 2 Option 3 Option 4

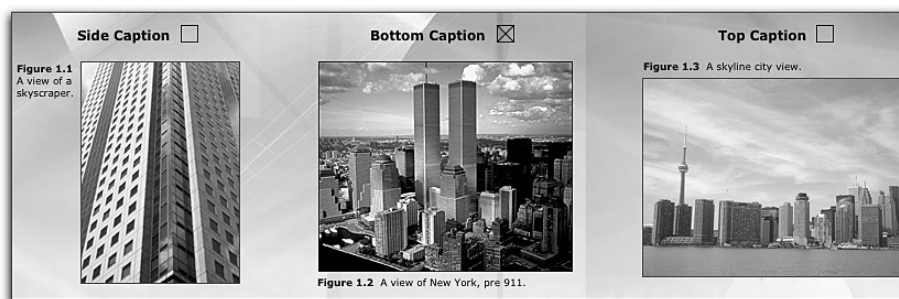
The purpose for these is that you will have books where you cannot use the same *Art Placement Option* for each type of art. For example, you may have a photo that requires a side caption and a figure that requires a bottom caption. This requires using both *Art Placement Option 1* and *Art Placement Option 2*. When selecting the “Option 2” button at the bottom of the page, it will take you to another page that looks very much the same, but will have *Art Placement Option 2* at the top.

Make sure the button is selected and then make your choices to this page. Fields such as “Job Number”, “Path to Art”, etc. will all bring information from the first page, however, the other fields specific to the art you are going to use will need to be filled out completely.

Each page should have different images showing in the caption positioning area so it feels like a different page. Or, you could possibly change the background slightly to give each new option a different feel. You’ll want all of the field information to be the same, so it should be subtle. I advise that each page needs to be unique enough to decipher between them, so you don’t accidentally start changing information on a page that shouldn’t be altered.

Caption Positioning

The caption positioning is user-defined by clicking “Side Caption”, “Bottom Caption”, or “Top Caption”. Below is an example of how the end result of the image and caption will import using Xtags. This makes it less confusing for the user to know how their image should import. You can only select one of the three because the script will calculate based on this information.



On the second page, *Art Placement Option 2*, you can select a different caption placement. It is a good practice to use different images on the other option pages.

The purpose of this selection is to keep from having to figure out the BL1, TL1, etc. placement that goes with picture and text boxes. The idea here is to select one of the caption placement options, and then let the scripts take over. You will have to know the width, depth, space to art, and other information about the image and caption. What you will not need to be concerned with is the coding of the Xtags fields, order of commas, etc. You will need that when setting up the database initially, but not when setting up the jobs.

Image Width and Depth

The *Image Width and Depth* section is the maximum art width and art depth which are represent in the #3 and #4 fields in the Xtags string. This also has a “Shrink to fit” button for the width and depth which is equivalent to the “?”. Here is an example of the generated string based on the field information:

```
<&pbu2(0 B,0,24p?,53p?,
```

Always be sure that the depth of the art and the depth of the caption do not exceed the total pasteboard depth.

Caption Width and Depth

The *Caption Width and Depth* is the maximum caption width and maximum caption depth allowable for the caption. The “Space from Art to Caption” is the white space between the text box and the image. This is handled with BL1, TL1, etc., spacing. I wrote this where it would be done in points to make things less complicated. The caption depth has an automatic shrink-to-fit built in. Here is an example showing where that information goes in the Xtags string:

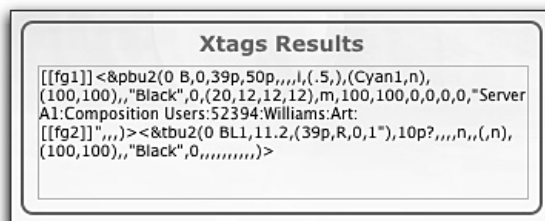
```
<&tbu2(0 BL1,12,24p,4p?,
```

The Relative button is if you have a caption that needs to be the same width as the the art when it shrinks-to-fit. Many designs require this. When this happens, the #3 field will look like:

```
<&tbu2(0 BL1,12,(24,R,0,0),4p,
```

It is significant to pay the most attention to these *Image Width and Depth* as well as the *Caption Width and Depth* sections because this is where you will be doing the majority of changes for each job.

Xtags Results



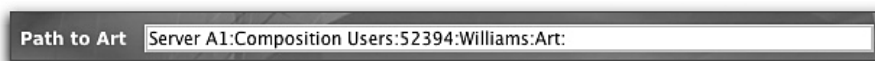
The Xtags Results displays the finished Xtags picture and text boxes for the image and captions. These are created in a *Calculation* within a field called “F1 Xtags Results” which will be set similar to this:

```
If ( F1 On_Off = "On" ; "<&pbu2(" & If (IsEmpty(F1 X Cdn); "0 B,"; F1 X
Cdn & " B,") &
If(IsEmpty(F1 Y Cdn);"0,"; F1 Y Cdn & ",") &
F1 Maximum Wdt & If ( F1 Shrink to Fit = "On"; "?" ; "" ) & "," &
F1 Maximum Dpt & If ( F1 Shrink to Fit 2 = "On"; "?" ; "" ) & "," &
```

I’m only going to show the first 4 commas of the picture box, but you should get the idea of how the *Calculation* is built. The field above that says “F1 X Cdn” is a field for the X Coordinate. I would only fill this in if you will be doing something unique with the positioning. The last field that will be put in before this calculation ends is for bringing in the path to the art. This will appear in the *Calculation* as:

& "&"" & **Path to Art**

This is read by the data in the “Path to Art” field:



The way to get the path to the art is by doing the following. This was mentioned earlier in the text, but instead of flipping back through the book, we’ll cover it once again:

1. Import an image from the folder you want.
2. Click on the image and select in QuarkXPress, *Edit:Copy Xtags Text*.
3. Create a text box and select *Edit:Paste*. The string will come in like this:

```
<&pbu2(0,0,383,231,,,n,,(n),(100),,n,,,m,,,,,,,"Server F1:Composition
Users:52394:Williams:Art:Chapter 03:52394_f0401.eps",,,)>
```
4. Copy only the highlighted area after the first quote to right before the word “Chapter”. Paste this information into the “Path to Art” field.

Advanced Options

At the bottom of the first page, there is a button named “Opt 1 Advanced”. When pressed, this button will take you to a new page that has the advanced Xtags options for the *Art Placement Option 1*.

The first page contains the fields that will need to be changed for each book. The *Advanced Options* are primarily for additional Xtags functions such as frames, runarounds, offsets, etc. I will go through part of this to give you an idea how this should be set up and used. It is a good idea to make sure you have a script that sets the default information upon making a new record. This ensures that the default information will be put into the Xtags strings.

Autopage Reference Options

Many users would want to have Autopage reference overrides built in. This allows them the ability to override the parameters if necessary. The way to achieve this is by first clicking the “Autopage Usage” on the first page, and

then adjusting the parameters here. In the “Autopage Placement” field, I have a value list defined for the following options that would create the following Autopage overrides:

Value List Option	Tag Override
Parameters	Do Not Alter
Top	Y=T
Bottom	Y=B
Left Top Side	X=L Y=T
Right Top Side	X=R Y=T
Left Bottom Side	X=L Y=B
Right Bottom Side	X=R Y=B
Top Outside	X=O Y=T
Bottom Outside	X=O Y=B

The VSB field would have the options to select 1, 2, or 3. The *Positioning* field would have either “Float” or “Inline”. If “Float” is selected, no override will happen since this is the default. When the script is run against the text, these overrides will end up in the [[AR ID#]] coding such as:

See Figure [[AR 5 X=L Y=B I=Y V=3]]1.5.

If the fields are left to the defaults, the coding will just be:

[[AR 5]]

Picture Box—Art Advanced (&pbu2)

The “Picture Box—Art Advanced (&pbu2)” section contains the remaining fields in the picture box tag beyond the first 4 fields. When making a new job in the database, the defaults will fill in. It is important to make sure you create a script that will put the defaults in each field. That will assure the user that if they do not touch the advanced options page that the right information will be entered into the Xtags strings. Here is a partial example of what is seen in this section:

Picture Box—Art Advanced (&pbu2)

Box - Skew, Angle, Flags, Placement, Color, Runaround

Box Skew: 0

Box Angle: 0

Flags: None

Bg Color: Black

Bg Shade: 0

Placement: Centered

☐ Runaround

Top:

Left:

Right:

Bottom:

As you can see in this example, the *Background Shade* is “0” and the *Placement* is “Centered”, therefore the tag will have the highlighted areas altered from their defaults:

```
<&pbu2(0 B,0,30p?,24p?,0,0,,n,,(n),(,100),,"Black",0,,c
```

Since the runaround is not selected, the “n” will go into the runaround field automatically. It's important to set this up with the error checking.

The way the *Calculation* works for the *Placement* is by filtering through each of the possibilities using a “Case” statement and matching on one of them. Here is how this would need to be written:

```
Case ( F1 Placement = "Default" ; "m,"; F1 Placement = "Centered" ; "c," ;  
      F1 Placement = "Expand/Shrink to Fit" ; "f,"; F1 Placement =  
      "Expand/Shrink Ratio"; "a,"; "m,")
```

This states that if the F1 Placement field = “Default”, put in an “m,” and continues through the other possibilities. Following the last semi-colon is another “m” which means if something else is in this field or nothing, choose “m,”.

I'm not going to go into detail for the Frame, Shade, and other settings, because they essentially work very much the same. Some fields have pull down menus where others you need to enter a number value. As you can see, there are a lot of possibilities here.

It is much easier for the user to select the options in the fields than to try to count commas. I will say it does take a considerable amount of time to build a database like this, but once it is completed, your composition department will be more cost-effective. If you do decide to build a database similar to this, my best suggestion is to make sure you do a lot of error checking and testing. It will pay off to have it near perfect the first time out.

Text Box—Caption Advanced (&tbu2)

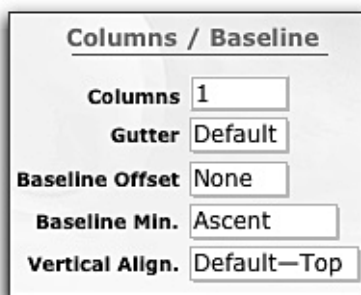
Text Box—Caption Advanced (&tbu2)			
Box - Skew, Angle, Flags, Placement, Color, Runaround			
Box Skew	0	<input type="checkbox"/> Runaround	<input type="checkbox"/> Text Inset
Box Angle	0	Top	Top
Flags	None	Left	Left
Bg Color	Black	Right	Right
Bg Shade	0	Bottom	Bottom

The “Text Box—Caption Advanced (&tbu2)” section contains the remaining fields in the text box tag beyond the first 4 fields. I'm only going to show the

first half of this. As you can see, the first part of this is set up exactly the same as the picture box tags, but the *Text Inset* is specific to text boxes.

The text inset amounts for Top, Left, Right, and Bottom, allow you the option for multiple insets. This even works if you are using Quark 4.1.1. Once you click the “Text Inset” button, it activates this and then you can add the amount for each field.

The frame and shading options are the same, but there are some columns and baseline information such as the offset, baseline minimum, etc. that need to be filled in as well.



I know this is a database I created and you can't use it off the shelf like other programs, but what I'm trying to accomplish here is that you can build something like this yourself, but you need to realize how much information will need to be included to accomplish this.

Exporting to Translation Table

This is one of the most important aspects because based on your scripts, field information, etc., you will need to write a script that takes all of that information and exports it to a BBEdit file to create the translation table.

This is achieved by writing a script that will go through the database and export information from the calculations. Here's an example of how the [[fg1]] and [[fg2]] tags will be built. I will start by instructing the script that:

```

If [Xtags Creator::F1 On_Off = "On"]
  If [Xtags Creator::F1 Figures Yes No = "On"]
    If [Xtags Creator::F1 Bottom Caption = "On" or Xtags
      Creator::F1 Bottom Caption = "On"]
      Set Field [Xtags Creator::TTL; Xtags Creator::TTL &
        "[[fg1]] " & Xtags Creator::F1 Xtags Results & "¶"
      Set Field [Xtags Creator::TTL; Xtags Creator::TTL &
        "[[fg2]] " & Xtags Creator::F1 Xtags Results Text & "¶"
    End If
  End If
End If

```


To break this down, this script is being instructed that:

1. If the “F1 On Off” field (The *Art Option Placement 1* field at the top of the page) is clicked, proceed to next step.

Art Placement Option 1 ☒

2. If the “F1 Figures Yes No” field is selected which is the Figures button next to Type of Art, proceed to next step.

Type of Art ☒ **Figures**

3. If the “F1 Bottom Caption” button is selected, then all three of these are proven “true”.

Bottom Caption ☒

4. The script will pull information from the “F1 Xtags Results” and the “F1 Xtags Results Text” fields and copy this data into the translation table.
5. This will pull the following information into the field named “TTL”:

```
[[fg1]] <&pbu2(0 B,0,30p?,24p?,0,0,,(n),(,100)),,"Black",0,,c,100,
100,0,0,0,0,"Server F1:Composition Users:52394:Williams:
[[fg2]] ",,)><&tbu2(0 BL1,14,30p,5p?,0,0,,(n),(,100)),,"Black",0,,,,,,,,,>
```

After all of the strings are compiled, a script titled *Translation Table Exported* will go to the layout where the Translation Table data is located, copy it out of the “TTL” field, then perform this AppleScript

```
"tell application \"BBEdit\"
    activate
    make new text document
    paste
    save text document 1 to file \"\" & Xtags Creator::Path to Art &
        \"Paging Support:\" & Xtags Creator::Author Last Name &
        \".ttl\"
end tell"
```

What this is doing is creating a new BBEEdit document, pasting the “TTL” field information and then saving the document into a Paging Support folder following the path to the art and in the Paging Support instead. This would be:

Server F1:Composition Users:52394:Williams:Paging Support:Williams.ttl

Database 2 — Style Sheet Generator

Design Selected: Standard 001

Style Sheet Generator

Book Title Breaking into Screenwriting **Edition** 2nd **Export Data** Xtags Only

Trim Size 7.25" x 9.5" **Chapter/Part Openers** Chapter Opener **Text/Leading** 11/13

Project Number 72004 **Author Last Name** Williams **ISBN (10 or 13)** 097620410X

Display Font Wexler Bold **Text Font** Lexington **Symmetrical Indent** Full Text Width

Style Sheets Required

Part Opener Elements:

- ☐ Part Number and Title Only
- ☐ Part with Text Elements

Chapter Opener Elements:

- ☐ Chapter Number and Title
- ☐ Chapter Intro Text
- ☐ Chapter Author and Affiliation
- ☐ Chapter Objectives

Main Level Heads:

- ☒ H1 Main Level Head
- ☒ H2 Main Level Head
- ☐ H3 Main Level Head
- ☐ Generic Head

Text Elements:

- ☐ Inline Elements (KT, URL, etc.)
- ☐ Text Elements (dialog, math, extract)
- ☐ Lists (BL, NL, LL, etc.)
- ☐ Sublists (NLBL, etc.)

Floating Elements:

- ☐ Margin Notes
- ☐ Figures
- ☐ Box 1 Elements
- ☐ Box 2 Elements
- ☐ Photos
- ☐ Footnotes
- ☐ Tables

Style 1 **Style 2** **Style 3**

Styles Page 2 **Figure Selector** **Box Selector** **Export to File**

I am not going to go into the *Style Sheet Generator* in the same amount of detail like I did the *Xtags Creator*. This is a very important database because its purpose is to make customized standard designs by generating style sheets based on Xtags Paragraph and Character Style Sheet information. This information is read not only by the checkboxes that are selected, but also by the fields at the top of the page. This also takes information from the *Figure Selector* and *Box Selector* pages allowing the user to customize which type of figure or box style theme they desire for their particular design.

Let's first look at the top of page where we have the following fields: *Book Title*, *Edition*, *Color*, *Page Size*, *Part / Section Openers*, *Text / Leading Size*, *Display font*, *Text Font*, and *Text Width Indent*. These are all critical to the success of the book. Let's take the second and third rows:

Trim Size 7.25" x 9.5" **Chapter/Part Openers** Chapter Opener **Text/Leading** 11/13

Display Font Wexler Bold **Text Font** Lexington **Symmetrical Indent** Full Text Width

All of these fields will have an impact on the style sheets. For example, if the text and leading is 10/12, each style sheet that isn't a title or needs special

font/leading requirements will need to have this called into the style sheet. The fonts are handled the same way. Let's look at a style sheet that is set up in the script. For example, here is the NL style with the leading, text size, and font highlighted:

```
Style Fields::Listing &
"@NL=[S\"\", \"NL\"<*><*>h\"just\"<*>kn0<*>kt(2,2)<*>ra0<*>rb0>
<*>d0<*>p(18,-18,0,13,3,0,g,\"U.S. English\")<*>t(13,2,\"1 \"18,0,\"1 \">
<P><s100><t0><h100><z11><k0><b0><cK><f\"Times-Roman\">¶"
```

This is very standard on how this is handled. Every text style will be predefined for an 11/13 font/leading size. So when “10.5/12.5” is selected, a script will be called to do the following:

```
"tell application \"BBEdit\"¶" &
  "activate¶" &
  If ( Style Fields::Text Leading Size = "10.5/12" ;
    "replace \"(<\\*p\\*(.+.+.+)(13)(.+.+)(<z11>)(.+.+Times)\"
      using \"\\*1,12.5\\*4<z10.5>\\*6\" searching in text 1
      of text document 1 options {search mode:grep, starting at
      top:true, wrap around:false, backwards:false, case
      sensitive:false, match words:false, extend selection:false}¶"
    ; "" ) &
  "end tell"
```

This is doing a “Grep” replacement that looks through the paragraph definitions and when it finds the “13” leading in the paragraph tags, it will swap this out to be “12.5”. When it locates the <z11> size tag, it changes this to <z10.5>. It is very straightforward, but you really need to make sure you do not delete any content while making these changes.

The same will be true with the font replacements. The font was written to always default to “Times”, but if you want to use a font such as “Giovanni”, you would need a script that goes through and does the following:

```
"tell application \"BBEdit\"¶" &
  "activate¶" &
  If ( Style Fields::Text Font = "Giovanni" ;
    "replace \"(Times-Roman)\" using \"Giovanni-Book\" searching in
      text 1 of text document 1 options {search mode:grep, starting
      at top:true, wrap around:false, backwards:false, case
      sensitive:false, match words:false, extend selection:false}¶" &
  "end tell"
```

To replace the italics and bold of this font, you would just have the replace lines do the following:

```
"replace \"(Times-Bold)\" using \"Giovanni-Bold\" searching in text 1 of text
document 1 options {search mode:grep, starting at top:true,
wrap around:false, backwards:false, case sensitive:false, match
words:false, extend selection:false}¶" &

"replace \"(Times-Italic)\" using \"Giovanni-BookItalic\" searching in text 1
of text document 1 options {search mode:grep, starting at
top:true, wrap around:false, backwards:false, case sensitive:false,
match words:false, extend selection:false}¶" ; "" ) &
```

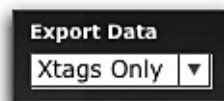
This is the best way to handle the fonts that I have found. The problem isn't that you couldn't just use the information in the field, but the fonts have a certain way they have to be identified that is more difficult for the user. It's easier to just put in "Giovanni" and then let the program realize the italic version needs to be "Giovanni-BookItalic" rather than having the pull down menu be full of all of these options.

To set this up properly, you should have a predetermined amount of fonts to put in the "Display Font" and "Text Font" categories. Since the idea is for standard designs, you want to give enough flexibility to make them unique, but not too much where it overcomplicates the process.

Indents

Another interesting field is the "Asymmetrical Indent" which lets you decide what type of indent is required on the text area if the main head straddles the margin area. This is where you can be tricky. If you are going to have the type always set right of the page where the straddle goes to the left, you could have all of the main text set on an indent.

If using Autopage, you would want the text area to be in its own text box while the heads are set in a text box straddling the entire margin area. A *field* controls this called "Export Data", which gives the two options of "Xtags Only" or "Autopage".



Depending on which is selected in the "Export Data" field, using "If Statements" will instruct the script if it has a "Asymmetrical Indent" of 7p and "Export Data" = "Autopage", to the styles with off of a 0p indent. If using "Xtags Only", the indent would start on 7p. Using the information in the fields is where the customization becomes easier for the end user.

Setting Style Sheets

If you wanted to take the time, in the “Style Sheets Required” section, it would be possible to make a style sheet checkbox for each style that is covered in the design. By doing this you open yourself up to interpretation problems where the user who checks these boxes starts wondering which style is needed, etc. I find it easier to handle the text elements, end matter elements, boxes, etc. by having a checkbox for those sections. Then, the majority of information that would normally be in those sections would be part of the design. Look at the screen capture for the “Text Elements” area.



By handling the text elements in one check box, you could have as many definitions expressed all in this one box. The Script would be written like this:

```
If [StyleGenerator::Text Elements = "On"]
    Perform Script ["Text Elements Combined"]
End If
```

What this will do is within the Text Element Script, all of the definitions for styles such as *text*, *dialog*, *display math*, *quotes*, *extracts*, *vignettes*, *poetry*, etc. would all be defined and added to the design. This would export the definitions of each style sheet to a text file that will be imported into the Quark template. You will want to have a line of generic text following the style sheet definition that will be imported. Here is an example of how the style sheet and the sample text will export to a text file. The sample text is highlighted.

```
@h1=[S"", "h1"]<*L><*h"H1"><*kn0><*kt(2,2)><*ra0><*rb0><*d0>
<*p(0,0,0,22,24,6,g,"U.S. English")><K><s100><t0><h100><z18>
<k0><b0><cK><f"Times-Bold">
@h1:[LC 1 M=40p A=2]]The Need for Automation[[SR h1 V=3 L=h1]]
@tx1=[S"", "tx1"]<*><*h"Standard"><*kn0><*kt(2,2)><*ra0><*rb0><*d0>
<*p(0,0,0,13,0,0,g,"U.S. English")><P><s100><t0><h100><z11>
<k0><b0><cK><f"Times-Roman">
@tx:[LC 1 M=30p]]Automation can be the difference between a
profitable company and one that reduces staff because of inefficiencies.
```

You will also want the AppleScript to place the styles in the order that you want the text to flow in the document. For example, I will put the book front

matter first, followed by the chapter number and title, objectives or vignette, or opening text. The order is important because it eliminates the need for later hand manipulation and produces a nearly finished design if scripted properly.

An argument could be made that the design would contain too many style sheets. I have found that it is much easier to ignore a style sheet than to have to add it later. I don't really feel that content that may not be used is a problem. The only time I feel that it may cause issues if it is content that requires the user to add Autopage codes or Xtags picture or text box string information. If it is just part of the text flow, it can be disregarded if not used.

Additional Features

You can customize a system like this as intricate as your workflow dictates. If you are an Autopage user, you may want to include layout changes that are scripted in and other automated possibilities. This database has features that are available by clicking on the buttons at the bottom right of the database:



These buttons include the following:

1. An additional page for styles which would include the end matter of the chapter such as the *Summary*, *Key Terms*, etc.
2. A *Figure Selector* which allows the user a chance to pick the style that the figures will be imported. This requires Xtags picture box and text box tagging.
3. A *Box Selector* allowing the user the opportunity to decide between different boxes styles. By clicking on the style wanted, the selected boxes will import. This also requires Xtags picture and text box tagging.
4. The *Export to File* button takes all of your choices and exports it to an Xtags text file that can be imported. You can also write an AppleScript that takes the exported file and import the Xtags directly into the QuarkXPress template. Additional scripting can have it run Autopage. The power here is based on what you need and the ability to understand scripting.

The other part that I wanted to point out is by selecting the “Style” buttons under the chapter opening image will tell the database to use one of three different styles.



If set up correctly, this provides the user the option of having three completely different appearances for the design. You can set this as high as you want to

depending on your needs. You could have twenty different design styles, but this requires extra scripting underneath.

What will need to be scripted will be that the design elements throughout will have to match the style. This makes it important to have scripts designed that will call out those elements based on which style is selected. Here's an example of how a script could be set up to interact with this:

```
If [StyleGenerator::Style Indicator = "Style 1"]
    Perform Script ["Style 1 Elements"]
Else If [StyleGenerator::Style Indicator = "Style 2"]
    Perform Script ["Style 2 Elements"]
Else If [StyleGenerator::Style Indicator = "Style 3"]
    Perform Script ["Style 3 Elements"]
End If
```

Where this is very helpful is within each of these "Style # Elements" scripts, there must be swapouts for the design elements. I suggest that the easiest way to accomplish this task is by making sure all of the elements appear different, but they still maintain the same width and depth.

A good example of this would be this box element. It will need to retain the same box characteristics for each design, but needs to have the same retain the theme of the "Style". Look at this example:

Box 3.3 Internet Home Businesses

With the onset of online shopping, auction companies, and the ease of use through wireless Internet, home business ventures are growing to be where some of the big money lies. Unlike the commercials that you see where it shows the entrepreneur sitting by his pool sipping a mixed drink, a home business does require a lot of extra time.

The bar at the top has an image that ties into the opening image keeping this style theme consistent throughout. For the second design option, a different style element would import which would change the box to have this appearance:

Box 3.3 Internet Home Businesses

With the onset of online shopping, auction companies, and the ease of use through wireless Internet, home business ventures are growing to be where some of the big money lies. Unlike the commercials that you see where it shows the entrepreneur sitting by his pool sipping a mixed drink, a home business does require a lot of extra time.

This would be accomplished by simply changing names in the replacements where the Xtags string would bring in “box_style1.eps”, and if “Style 2” was selected, it would replace for “box_style2.eps”. Consistency in naming is very important to reduce the chance for errors.

Database Production Benefits

In the past couple of years, I've found the use of FileMaker Pro databases to be a very efficient practice to achieve automation. It is so user-friendly and allows you a single location to store all scripts associated with the project and being able to call them in with the use of the database field content.

This is why I suggest to anyone who wants to get the most value out of their prepress dollar to consider building their own databases for their production departments. They can be server based and password protected to keep the scripting secure. That is one of the problems I find with just using AppleScripts among many users is that anyone can go in and change these and you might end up with different versions on your server.

In order to really get the value out of a database, it is important to have time to commit to one. You can get started and have something useable, but it is important to be able to stay on it until it is completed. The best way to do this is come up with an idea, draw out how you plan to get there, and then find gaps in the production cycles and start building it. Start slow, get one thing working, and then move on to the next. Don't make the mistake of having 30 uncompleted things in one database. That will end up being troublesome later. My opinion is that anytime you can have a group of people working in a program that is customized to the needs of your company, profitability will be the end result.

Xtags 7.3 and Beyond...

I really anticipate that you will have picked up many new techniques in this book that can be applied to your current workflow. It is in your best interest to practice the techniques shown within this book whenever time permits. Try to work with taking exported Xtags content and repurposing for the web or XML using BBEdit. I really believe this is the direction more textbook publishers will be taking and you can really be ahead of others by customizing your workflow and finding faster methods for repurposing. You will find that this can really be job security in the changing publishing market.

Xtags 7.3 Overview

Xtags is a software that is constantly expanding and growing. Some major advancements reside in the latest Xtags update supporting Quark 7. I believe it's one of the most important Quark updates ever. I plan on having another edition of *Xtags Maximized* next year that will dive deeply into the new table tags, feature more box examples, as well as things that may have been omitted. At press time, Em Software just released Xtags 7.3 for Quark 7. I want to touch briefly on what I have noticed so far with this new edition.

Table Tags

One of the latest features is the long-awaited table tags that can create tables similar to Text Box (&tbu) and Picture Box (&pbu) tags. These are featured in Xtags 6.3 and Xtags 7.3. These are very advanced tags, which also work in combination with Autopage to speed up the paging process even further. The table tags will be handled with unique tags. Here is a brief overview of these:

Unanchored tables <&tsu(x, y, width, height, # of columns, # of rows,
column widths, table angle, flags, runaround, etc.)>

Unanchored ending <&tse>

218 *Xtags Maximized*

Anchored tables	<&ts(width, height, # of columns, # of rows, column width, flags, text align, frame width, etc.)>
Table row start	<&trs(type, height, bg color, bg shade, etc.)>
Table row end	<&tre>
Table cell text start	<&tcs(width, height, horiz span, vert span, etc.)>
Table cell pictures	<&tcp(width, height, horiz span, vert span, etc.)>

Many of the parameters within these tags are the same as the text and picture box tags. These tags are very advanced and work best with Quark 6.5.2 or later. The table tags are not written in detail in this book because they were released less than a week prior to this book going to print and I haven't had enough time to work with them.

I will show one example of something very basic. This is the code that I put together to bring in a basic 2 row, 2 column table with some text in it:

```
<&tsu(0,0,165,60,2,3,0,,,2,,20,)>
<&trs(n,36)>
<&tcs>@tbl:This is <B>Cell 1<&tcs>@tbl:This is <B>Cell 2<&tre>
<&trs(n,36)>
<&tcs>@tbl:This is <B>Cell 3<&tcs>@tbl:This is <B>Cell 4<&tre>
<&trs(n,36)>
<&tcs>@tbl:This is <B>Cell 5<&tcs>@tbl:This is <B>Cell 6<&tre>
<&tse>
```

The table is broken down as:

<&tsu> line is broken down as: x (0), y (0), width (165), height (60), number of rows (2), number of columns (3), column widths (0), frame width (2), frame shade (20%). Default values for the rest of the table.

The <&tcs> (columns) were left for all default values.

The <&trs> (rows) were given a “n” to indicate that they were for normal rows where an “h” would have indicated a “table header row” and an “f” would specify “table footer row”.

The imported result looks like this:

This is Cell 1	This is Cell 2
This is Cell 3	This is Cell 4
This is Cell 5	This is Cell 6

There are so many options in the table tags that an entire chapter will need to be set aside to cover these thoroughly in the next edition.

PSD Importing

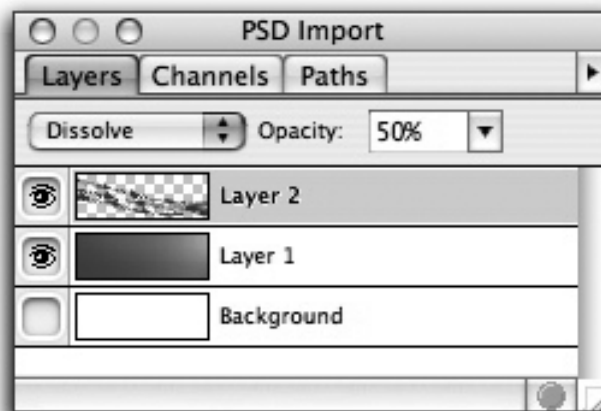
Quark 7 has made PSD importing possible. Xtags also has the capability to import these as well without any special handling outside of calling in the piece of art such as:

```
<&pbu2(0 B,0,289,210.5,,,n,,(n),(,100)),n,,m,,,,,  
,, "Macintosh HD:Xtags Book:Desktop:fg09_033.psd",,,)>
```

From working with these images during import, certain filters used in Adobe Photoshop are not supported by the PSD Import resulting in the image being handled as a composite without the layers being active. If you import an image with Xtags and this occurs, an error message will appear stating:

This image contains layers with effects or color adjustment modes that PSD Import currently does not support. PSD Import will use the embedded composite image.

At this point you could either change your image to be compatible or just work with it as is. At a minimum, the PSD file is still available in the document. When a PSD image does import without becoming a composite image, the palette offers the ability to work with image *layers*, *channels*, and *paths* as seen in the screen capture below.

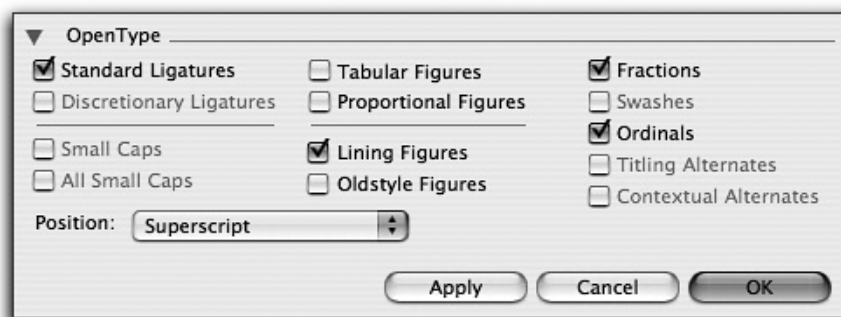


It also offers the ability to work with opacity on separate layers as well as effects such as dissolve, multiply, overlay, etc. The ability to work with PSD files is an excellent feature within Quark 7 that is worth upgrading to this version immediately. The fact that Xtags already supports the import makes this a very important update.

To the best of my knowledge, there isn't a way to turn off the different layers of a PSD file upon import.

OpenType® Support

Quark 7 now supports *OpenType* fonts (.otf fonts), which have special characters including ligatures, substitute versions of characters, swashes, and other type options. Here is the *OpenType* addition in the *Character Attributes* dialog:



Xtags has the capability of calling out multiple selections in one tag. The options are available in the character style sheets. To override a setting using Xtags to include OpenType, the following coding structure would apply:

```
<o("liga","ordn","frac")> or <o("smcp","lnum")>
```

The coding structure for most of these OpenType options are as follows:

liga	(ligatures)	onum	(oldstyle figures)
dlig	(discretionary ligatures)	pnum	(proportional figures)
ordn	(ordinals)	lnum	(lining figures)
titl	(titling alternatives)	tnum	(tabular figures)
frac	(fractions)	numr	(numerator position)
swsh	(swashes)	sup	(superscript position)
c2sc	(all small caps)	subs	(subscript position)
smcp	(small caps)	dnom	(denominator position)

An example of *OpenType* would be if you were importing fractions. If the coding department typed the following:

```
@dm:The fractions are 1/2, 1/4, and 3/4.
```

The text would import as it looks in the coded file:

The fractions are 1/2, 1/4, and 3/4.

But, if the typist added the *OpenType* code at the start of the paragraph:

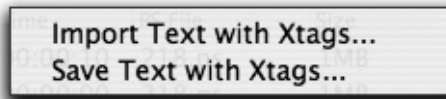
```
@dm:<o("frac")>The fractions are 1/2, 1/4, and 3/4.
```

The text would import using the substituted *OpenType* fractions:

The fractions are ½, ¼ and ¾.

Menu Differences

A few differences exist in the menus in the Xtags 7.3 version. Instead of *Get Text with Xtags*, Quark 7 introduces *Import Text with Xtags*. This matches how Quark 7 no longer uses *Get Text*, but rather *Import Text/Picture*.



The “Edit” menu also has *Paste with Xtags* and *Copy with Xtags*, which are slightly different from the “Copy Xtags Text” and “Paste Xtags Text” from all previous Xtags versions.



Insert Special Characters

Under *Utilities:Insert Character* are two options: “Special (nonbreaking)” and “Special”. This contains practically any spacing you can imagine including thin space, hair space, flexible space, etc. To have to go to this menu each time in Quark would be time consuming. With that said, these are available as Xtags markup codes. The coding structure for these are as follows:

Special

Em Space	<\m>	Em Dash	<_>
En Space	<\e>	Discretionary Hyphen	<\h>
3-per Em Space	<\5>	Indent Here (Hang)	<\i>
4-per Em Space	<\\$>	Discretionary New Line	<\d>
6-per Em Space	<\!^>	Punctuation Space	<\p>
Thin Space	<\[>	Figure Space	<\8>
Hair Space	<\{>	Flexible Space	<\f>

Special (nonbreaking)

Em Space	<\!m>	Flexible Space	<\!f>
En Space	<\!e>	Figure Space	<\!8>
3-per-Em-space	<\!5>	Punctuation Space	<\!p>
4-per-Em-space	<\!\$>	Standard Space	<\!s>
6-per-Em-space	<\!^>	Em Dash	<\!_>
Thin Space	<\![>	Hyphen	<\!->
Hair Space	<\!{>		

Many of the codes have previously been part of the Xtags structure, but now that these are in a Quark pull down menu, the coding of these will be even more

necessary. Notice that “en space” is now “<\e>”. In previous versions of Quark, this was “<\f>”. Many of these codes will only work with Quark 7 and higher.

One unique aspect of using these “Special” and “Special (nonbreaking)” characters is that when the *Invisibles* are “on” they each have their own character that is visible in the text. These are all searchable through the *Find/Change* window.

Paragraph and Character Tag Differences

As shown in Chapter 2, the Paragraph Styles tags normally appear as:

```
@h3=[S", "h3"]<*L><*h"P3"><*kn0><*kt(2,2)><*ra(1,"Solid",K,60,0,0,9)><*rb0><*d0><*p(0,0,0,9,0,0,g,"U.S. English")><P><s100><t0><h100><z8><k0><b0><cK><f"Bembo">
```

With the new Quark 7 features of *Opacity*, *Enable Ligatures*, *Language*, and *OpenType*, these tags are now expanded shown highlighted below:

```
@h3=[S", "h3"]<*L><*h"P3"><*kn0><*kt(2,2)><*ra(1,"Solid",K,60,50,0,0,9)><*rb0><*d0><*p(0,0,0,9,0,0,g,"U.S. English")><P><s100><p75><t0><h100><z8><k0><b0><cK><f"Bembo"><n0><o("dlig","smcp")><G1>
```

The first change above is the “50” for 50% opacity in the 5th field of the *Rule Above* (<*ra>) tag, which also works with the *Rule Below* (<*rb>) tag. The next addition is the <p75> *Text Opacity* tag for 75% opacity.

The <n0> tag is for the *Language* option in the Character Attributes. I have “U.S. English” and “None” as options in my version. The “U.S. English” gives the <n0> result, where <n254> works with “None”.

The “<o("dlig","smcp")>” tag is for *OpenType* (see page 220), and the <G1> is for the *Enable Ligatures* option in Character Attributes. “<G1>” for selecting *Enable Ligatures* and “<G0>” when turned off.

Text Opacity



One new important feature in QuarkXPress 7 is that opacity has been added for boxes, pictures, text, lines, rules, and other features. Xtags supports the opacity for text and rules. At press time, opacity doesn't seem to work with Xtags on boxes, lines, pictures, etc.

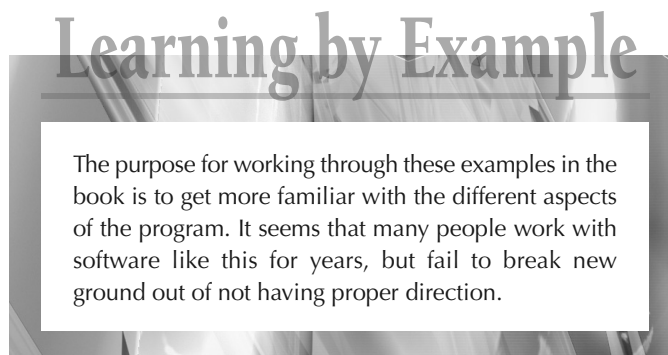
The text opacity uses a <p%> opacity text tag that works like this:

```
@Title:<p40>San Francisco<p$>
```

This will make the type “San Francisco” have a 40% opacity and the image is transparent below. Prior to this edition, this would have to be completed in a program like Adobe Photoshop.

Example Using Text Opacity and PSD Import

I thought it would be a good idea to show a few new features in one example. In this box, we are importing a PSD image with active layers and bringing in the type using text and rule opacity.



This also illustrates how you can use a background to create a unique border. Using certain image effects, this can take on many different variations. To get this result:

1. The “Learning by Example” box is standard, but will need 35% opacity on the rule and the type. This can be added in the style sheets, but for this example, I’m showing this as an override highlighted below:

```
<*rb(2,"Solid",K,70,35,0,0,6)><p35>
```

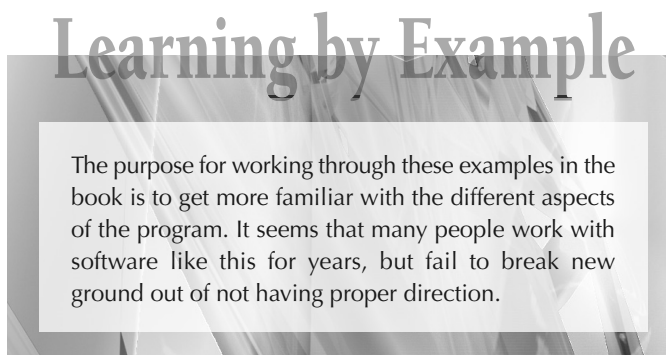
2. The second box is the text box below. It will have a Black “0%” background to mask off the image behind it. This is very standard.
3. The picture box will import the image with the “Send to Back” *flag* active. The relative placement of “(35p,R,3p,0)” positions this correctly where 12 points are to the left, right, and bottom, giving it a unique border effect that will look different depending on the depth.

Since this is a “PSD” file, there are many options to choose from because the layers are all editable inside of Quark. Each layer displays the active Photoshop filter. These can be changed inside of Quark 7 with this latest update allowing you to not have to jump back and forth between Quark 7 and the art program.

The code for this entire box would be:

```
<&tbu2(1p B,1,19p,5p?,,,,n,,(n),(,100),,n,,,,,,>@Title:<p35>
<*rb(2,"Solid",K,70,35,0,0,6)>Learning by Example<&te><&tbu2(0 BL1,
8,19p,35p?,,,,,(n),(,100),,K,20,,,,(12,12,9,12),, ,,,>Text Here<&te>
<&pbu2(-12 TL1,-2p1,21p,(35p,R,3p,0),,k,n,,(n),(,100),,n,,m,,
,,,,,"Macintosh HD:Users:Xtags Book:0700.psd",,,)><&g(3,2,1)>
```

By turning off several layers or altering the effects, the border effect can look different each time you use it. The text box opacity could also be changed in Quark to have a 75% opacity. Changing the opacity of the text box is currently not an option with Xtags, but I'm very hopeful it will show up in a future update. If altered to 75%, it would give this appearance:



In Closing

While there are features I would like to see added to a future update of Xtags such as box, line, and picture opacity, Quark blends, applying the new Drop Shadow feature directly to imported images, and change case, I can get so much mileage on what is already available. Workarounds for drop shadows and blends are shown in Chapter 4.

I'm also working with the Xtags version for Adobe InDesign to see the differences within the program. As mentioned earlier, if enough interest is out there, I may write a book focusing on Xtags for InDesign. I would like to see *Copy Xtags Text* and *Save Text with Xtags* added to the InDesign version. These are invaluable for repurposing content and figuring out tags. With so many new additions in Quark 7, I still find myself more of a QuarkXPress fan.

I anticipate that this book helped you find new methods for using Xtags and automating your workflows. I would expect that you have battled some troublesome boxes in the past that weren't shown here. If you encounter a unique box or image/caption combination that you just can't figure out, email me at highvolume@mchsi.com and I'll see if I can write code for it. If it is something I haven't seen before, it may be featured in the next edition. I will give credit in the book for the discovery to the person who sends this to me. I look forward to your feedback.

Index

- Additional Height Options, 76
- Adobe Illustrator*, 58, 86, 87, 101, 195
- Adobe InDesign*, 9, 224
- Adobe Photoshop*, 90, 101, 195, 219
- Align with Text, 125
- Altering Boxes to New Style, 154
- Anchored Alignment, 65
- Anchored Lines, 65–66
- Anchored Picture Boxes, 56–58, 75, 170–171
- Anchored Text Boxes, 15, 56–60, 75, 170, 174
- Angle, 57, 62
- AppleScript*, 9, 12, 16, 18–20, 31–32, 35, 50–51, 64, 86, 117, 126, 142–146, 148, 161, 175, 178–179, 190
 - FileMaker Pro, 192, 209, 213, 216
 - Get and Save Text, 15
- Arrow Type, 63, 65
- Ascent, 125, 51
- Automating Using Databases, 106
- Autopage*
 - Adding Codes with “Grep”, 152
 - Anchoring the H1 Side Head Instead of Side Art, 174
 - AutoTag Export Conversion Status, 178
 - Batch Export, 15, 38, 159, 175–178, 193–194
 - C=O Caption Option, 71–72
 - Exporting, 15
 - Handling After Import, 172
 - Horizontal Alignment, 54, 160–161, 163, 167–168
 - Importing, 15
 - I=T Option, 167
 - Left and Right Art Placement Option, 163
 - Left and Right Table Placement, 164
 - Oversized Two-Column Inline Art, 172
 - Paginate Window, 158–159
 - Quality Control, 6
 - Reference Options, 205
 - Rotated Tables, 162
 - Short Line Elimination, 7
 - Side Art – Using Reference, 160
 - Side Element Placement, 161
 - T=E Code, 67, 101, 115, 166
 - Text Wraps (Runaround), 168
 - X=O Option, 167
- Background Color, 46
- Background Shade, 46
- Baseline Minimum, 51
- Baseline Shift, 25
- BBEdit*, 17, 18, 31, 138–140, 142–144, 175
 - Find and Replace, 175, 180, 182
 - Grep*, 20, 22, 31, 138–140, 142–144, 148–154, 175, 178, 180, 182–185, 190, 193, 197, 211
 - Operators, 139
 - Search Results, 38, 51
 - Subpatterns, 138
- Bottom Caption, 54

226 *Xtags Maximized*

- Boxangle, 43, 133
- Box Name, 50
- Boxskew, 43, 133
- Box with 3 Separate Parts, 131
- Bringing in Multi-Piece Figures Together, 80
- Caption Positioning, 51
- Caption Top, Source Line Bottom, 85
- Changing the Inset (for Autopage 5.8), 32
- Character Information in the Style Sheet
 - Definition, 24
- Character Styles, 2, 7, 25, 26
- Coded File, 118
- Collect For Output, 39
- Color, 20–22, 25–26, 42, 65
- Combining Elements, 109
- Complex Box with Seven Xtags Strings, 117
- Complicated Boxes
 - Art, 115
 - Multiple Elements, 128
 - Box with Rotated Text, 119
 - Figure Caption Usage, 82
- Copy and Paste Xtags, 16
- Copy with Xtags, 221
- Copy Xtags Text, 12–13, 16–17, 32, 36, 38–39, 42, 71, 97, 126, 142–143, 164, 204
- Creating Boxes with Xtags, 101
- Custom Publishing Using Xtags, 141
- Delimiters, 13
- Don't Export Style Sheet Definitions, 11
- Drop Caps, 22
- Error Appearing in the Last Space, 39
- Error Code (32768), 39
- Expand or Shrink-to-Fit with Graduated Screen, 130
- FileMaker Pro*, 19, 35, 147, 192–193, 197, 216
 - Additional Features, 214
 - Advanced Options, 205
 - Art Placement Options, 201
 - Calculated AppleScript, 198–199
 - Caption Positioning, 202
 - Caption Width and Depth, 203
 - Database Production Benefits, 216
 - Exporting to Translation Table, 208
 - Image Width and Depth, 203
 - Importance of Databases, 194
 - Indents, 212
 - Level of Understanding, 194
 - Native AppleScript, 198
 - Perform AppleScript 198
 - Scripting Option, 197
 - Setting Style Sheets, 213
 - Setting up the Database, 198
 - Type of Art, 201
- First Baseline, 50–51, 87, 133
- Fit-to-Height, Fit-to-Width, 73–74, 95–96
- Flags, 44, 62, 64, 103
- Font Definition, 26
- Font Usage, 182
- Forced Anchored Leading, 75–76
- Forced Return, 38
- Frame Color, 45
- Frame Gap, 45–46
- Frame Height, 45
- Frame Shade, 45
- Frame Shape
 - concave, 45
 - convex, 45
 - oval, 45, 109
 - straight, 45
- Frame Style, 46
- Frame Width, 45
- Get Text with Xtags, 13–16, 18, 32, 158
- Grouping, 42, 56, 76, 78, 81, 83, 113, 163
- Headers and Contents, 7
- Height, 42
- Horizontal and Vertical Scale, 25
- HTML, 175, 184, 188–189, 192
- Hyperlinks, 192
- Hyphenation and Justification (H&Js), 19
- Indented Relative Placement, 69
- Interparagraph Max
- Inline Text Elements, 113
- Insert Character, 221

- Keep Lines Together, 20
- Keep with Next, 20
- Kerning, 25
- Labels Below Art (Alphas), 79
- Layer Name, 50, 64
- Line Modify, 63
- Line Name, 64
- Line Type, 63
- MacPerl, 20, 22, 32, 117, 126, 138, 161, 169–170
- Master Guides, 176
- Master Pages, 37
- MathType, 28, 56, 75, 125
- Menu Differences in Xtags 7.3, 221
- Minimum Height, 76
- Modify Window, 50
- Multi-Columns, 111
- Multiple Inset, 32, 119, 130, 208
- Offset x, 48, 68
- Offset y, 48, 68
- Omit Default Elements in Xtags List Tags, 12
- Origin Tags, 97–100
- Output separate tags (<I> vs. <BI>), 13
- Oval Text Box with Offset Shadow, 127
- Paragraph Alignment, 19
- Paragraph Attributes, 125
- Paragraph Parameters, 23
- Paragraph Style Sheets, 26–28, 30, 179–180
- Paste with Xtags, 221
- Paste Xtags Text, 12–13, 16–18, 42, 221
- Pasteboard, 18, 78, 83, 85
 - Importing on, 61
- Perl, 17
- Picture Box Using Skew, 123
- Picture Pathname, 49
- Picture Usage, 100
- Pixangle, 49
- Pixskew, 49
- Placement, 47–48
- PowerMath, 75
- Profitability, 3
- PSD Files, 219
- QuarkXPress 4.1.1, 1, 32, 38, 47, 137, 208
- QuarkXPress 6.5, 1, 32, 47, 137
- QuarkXPress 7, 1, 137, 217
 - Character Attributes, 222
 - Enable Ligatures, 222
 - Menu Differences, 221
 - OpenType Support, 220
 - Paragraph Attributes, 222
 - PSD Importing, 219, 223
 - Rule Opacity, 222
 - Special Characters, 221–222
 - Table Tags, 217–218
 - Text Opacity, 222–223
- Quark Blends, 101
- Relative Caption Placement, 62, 68–69, 72, 74, 78, 81–83, 87, 89, 91, 93–94, 104–105, 114, 119–120, 129–130, 132–133, 135–136, 147, 172
- Relative Double Border, 84
- Report Errors, 46
- Repurposing Content, 159
 - Closing the File, 188
 - Floating Element Markers, 189
 - Heads and Run-in Heads, 186
 - Images, 186
 - Know the Content, 188
 - More Intricate Coding, 183
 - Searching Through the File, 187
- Rotated Solid Shadow on Image, 89
- Rounded Corner Box with Unsupported Shape, 106
- Rule Above and Below, 21
- Rule Around Picture and Caption, 77
- Runaround, 44, 62–63, 125
- Save Text with Xtags, 13, 15
- Scale x, 48
- Scale y, 48
- Screened Caption Below Image, 94
- Screened Caption Under Image, 93
- Select QXP Doc in Export Hierarchy, 177
- Setting the Preferences, 11
- Shading, 25, 63
- Shadow on Images Shrinking-to-Fit Both Dimensions, 90

228 *Xtags Maximized*

Shrink to Fit Capabilities, 36, 43–44,
47–48, 61, 67–68, 74, 77, 91, 101,
103, 112, 114, 117, 130, 132–133

Side Caption

Bottom Left, 54
Bottom Right, 55
Top Left, 52
Top Right, 52

Side Element with 5 Grouped Boxes, 82

Side Margin Box Expandable, 121

Side Margin Box with a Vertical Line, 110

Single-Column Box, 101, 103

Sizing Options, 76

Skew, 57

Space Between Rule and Figure, 76

Spacing Tags, 221

Stray Colors, 39–40

Style Sheets

Definition, 19
Overrides, 26
Tag Attributes, 18

Table with Art Handling, 114

Tabs, 23–24, 30, 33, 162, 187

Templates, 34–35

Text Align, 56

Text Outset 47

Top Caption, 53

Tracking, 25

Translation Tables, 12, 14, 17, 22, 28,
32–33

Codes, 35

Format, 35

First Character Delimiter, 33

French Quotes vs. Brackets, 32

Hangs, 33

Start Tag, 33

Testing the Codes, 36

Troubleshooting Tips, 38

Type Size, 25

Type Style, 24

Unanchored Lines, 61

Unanchored Picture Box Tags, 43

Unanchored Text Box Tags, 50

Unique Box Treatment Using Skew and
Angle, 133

Unsupported Shape Box, 107

Using all the Placements in One
Element, 135

Using Lines for Top and Bottom of Box, 124

Using Xtags for Design Element in Box, 134

Vertical Alignment, 51, 130

Vertical and Horizontal Mixed Relative
Placement, 70

Vertical Relative Placement, 70

Width, 43, 62

Working with Blends in Captions, 86

Working with Figures, 76

Working with Multi-Column Text
Boxes, 111

XML, 15, 39, 101, 107, 115, 157, 159, 175,
177–179, 188, 192, 218

XMLxt, 5, 15, 39, 159, 175

Xtags Export Issue, 31

Xtags Fields, 41

Xtags Issues, 42

Xtags User Guide, 8–9, 11, 38, 43