



Xtags User's Guide

March 2008

Contact Information

Em Software, Inc.
503 Belleview Blvd.
Steubenville, Ohio 43952 USA
web www.emsoftware.com
email support@emsoftware.com or sales@emsoftware.com
vox (+1) 740 284 1010
fax (+1) 740 284 1210

Copyrights, Trademarks & Legal Notices

This manual and software are Copyright © 1990–2008, Em Software, Inc. All rights reserved. Xtags, Xdata, InData, Xcatalog, and InCatalog are trademarks of Em Software.

Portions of the software (the XTensions glue code) are Copyright © 1990–2008, Quark, Inc.

Adobe InDesign, Adobe Photoshop, and Adobe Illustrator are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

All other trademarks and registered trademarks are property of their respective holders.

Notice from Quark, Inc.: **THIS SOFTWARE PACKAGE HAS NOT BEEN WRITTEN, REVIEWED, OR TESTED BY QUARK, OR THEIR LICENSORS. QUARK AND THEIR LICENSORS MAKE NO WARRANTIES, EITHER EXPRESS OR IMPLIED, REGARDING THE ENCLOSED SOFTWARE PACKAGE, ITS MERCHANTABILITY, OR IT'S FITNESS FOR ANY PARTICULAR PURPOSE. QUARK AND THEIR LICENSORS DISCLAIM ALL WARRANTIES AND CONDITIONS RELATING TO THE SOFTWARE PACKAGE WHETHER EXPRESS, IMPLIED, OR COLLATERAL, INCLUDING, WITHOUT LIMITATION, ANY WARRANTY OF NON-INFRINGEMENT, COMPATIBILITY, OR THAT THE SOFTWARE IS ERROR FREE OR THAT ERRORS CAN OR WILL BE CORRECTED.**

IN NO EVENT SHALL QUARK OR THEIR LICENSORS BE LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL, CONSEQUENTIAL, OR PUNITIVE DAMAGES, INCLUDING, BUT NOT LIMITED TO, ANY LOST PROFITS, LOST TIME, LOST SAVINGS, LOST DATA, LOST FEES, OR EXPENSES OF ANY KIND ARISING FROM INSTALLATION OR USE OF THE SOFTWARE OR ACCOMPANYING DOCUMENTATION IN ANY MANNER, HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY. IF, NOTWITHSTANDING THE FOREGOING, QUARK AND/OR THEIR LICENSORS ARE FOUND TO HAVE LIABILITY RELATING TO THIS SOFTWARE PACKAGE, SUCH LIABILITY SHALL BE LIMITED TO THE FEES PAID BY END USER TO QUARK, IF ANY, WITHIN THE ONE -YEAR PERIOD PRECEDING THE CLAIM, FOR THE LICENSE OF THE SPECIFIC QUARK PRODUCTS (EXCLUDING ANY THIRD-PARTY COMPONENTS ADDED BY END USER OR ANY THIRD PARTY, INCLUDING DEVELOPER OR AN INTEGRATOR), OR THE LOWEST AMOUNT UNDER APPLICABLE LAW, WHICHEVER IS LESS. THESE LIMITATIONS WILL APPLY EVEN IF QUARK AND/OR THEIR LICENSORS HAVE BEEN ADVISED OF SUCH POSSIBLE DAMAGES.

Contents

BEGINNING WITH XTAGS5

What is Xtags?.....	5
About This Manual	5
Before Installing Xtags.	5
What you should know about your computer	5
What you should know about QuarkXPress/InDesign	5
Platform-specific issues	6
Please download the latest version.....	6
Understanding Xtags version numbers.....	6
Installing Xtags.....	7
For QuarkXPress	7
For InDesign	7
Installing InFlow.....	7
Un-installing Xtags.....	7
For QuarkXPress.....	7
For InDesign	7
Personalizing Your Copy of Xtags	7

USING XTAGS INTERACTIVELY9

Conventions Used in the Documentation	9
<i>Example 2.1</i>	9
Xtags Preferences.....	9
Importing Text With Xtags.....	11
Saving Text with Xtags.....	11
Copying and Pasting Xtags Text.....	12

LEARNING XTAGS BASICS 13

General Information	13
<i>Example 3.1</i>	13
Control characters.....	13
Word spaces within tags.....	13
Splitting long tag sequences.....	14
Using default settings.....	14
Tag Parameters—The Heart of Xtags.....	14
Tags with single parameters.....	14
Tags with multiple parameters	15

Tags with multiple parameters and sub-list parameters.....	15
Omitted parameters	15
<i>Example 3.2</i>	15
Tips for Constructing and Debugging Tags	17
<i>Example 3.3</i>	17

SETTING CHARACTER ATTRIBUTES 18

Reset Attributes Tags	18
Character Face Tags	18
<i>Example 4.1</i>	18
Character Size and Font Tags	19
<i>Example 4.2</i>	19
Character Color, Shade, and Opacity Tags.....	19
<i>Example 4.3</i>	20
Character Scaling, Kerning, and Tracking Tags.....	20
<i>Example 4.4</i>	20
Character Baseline Shift Tags.....	21
<i>Example 4.5</i>	21
Character Language Tag.....	21
Character Ligatures Tag.....	22
Special Character Tags.....	22
Character Set Encoding Tag and XPress Tags Version Tag.....	22

SETTING PARAGRAPH ATTRIBUTES 24

Paragraph Alignment Tags.....	24
<i>Example 5.1</i>	24
Paragraph Basic Settings Tag.....	24
<i>Example 5.2</i>	25
Paragraph Tab Settings Tag	25
<i>Example 5.3</i>	26
Paragraph Hyphenation and Justification Tag.....	26
Paragraph Rules Tags.....	26
<i>Example 5.4</i>	27
Paragraph Drop Caps Tag.....	27
<i>Example 5.5</i>	27
Paragraph “Keep Together” and “Keep With Next” Tags	28
<i>Example 5.6</i>	29

DEFINING AND APPLYING STYLE SHEETS 30

Defining Paragraph Style Sheet Tags	30
Defining Character Style Sheet Tags.....	30
Applying Style Sheet Tags	30
<i>Example 6.1</i>	31

CREATING TEXT, PICTURE, NONE, AND LINE BOXES..... 33

Box Creation using Basic Parameters.....33

Tags for Text Boxes 33
 Parameters for anchored and unanchored text boxes 34
Example 7.1 36
Example 7.2 37
 Tags for Picture Boxes..... 37
 Parameters for anchored and unanchored picture boxes 38
 Picture handling 40
Example 7.3 40
 Conditional picture import..... 41
Example 7.9 41
 Extended picture path parameters..... 41
 Tags for None Boxes 42
 Tags for Lines 42
 Parameters for anchored line tags..... 42
 Parameters for unanchored line tags..... 43
Example 7.4 44

Expanded Box Creation Parameters.....44

Absolute and Relative Box Placement 44
 Automatic Box Resizing 45
 Box width parameter 45
 Box height parameter..... 46
 Shrink-to-fit text boxes..... 46
Example 7.5 47
 Shrink-to-fit picture boxes..... 47
Example 7.6 47
 Picture Box Runaround Types (unanchored only)..... 48
 Item runaround 48
 Box Frame Specifications 48
 Frame width 48
 Frame color 49
 Frame shade 49
Example 7.7 49
 Box Background Opacity and Blends 50
 Background color 50
 Background shade 50
Example 7.8 50
 Text Insets and Outsets..... 50

Grouping Tags.....50

Grouping Unanchored Boxes 50
Example 7.10 51
 Set/Clear Relative Origin Tags..... 51
Example 7.11 51

CREATING TABLES 52

Tags for Tables 52
 Parameters for anchored and unanchored table boxes..... 53
Example 8.1 54
 Tags for Table Rows 56
 Tags for Table Cells 57
Example 8.2 57

APPLYING MASTER PAGES..... 59

Example 9.1 59

WORKING WITH TRANSLATION TABLES AND MACROS 60

Translation Tables60

Using the Translation Table Tag..... 60
 Translation Table Format..... 60
 Translation specifications 61
Example 10.1 61
 Adding Entries to a Translation Table 62
 Turning Off Tag Interpretation..... 63

Macros63

Macro Definition and Invocation Tags 63
 Separator-specifying macro definition tag 64
Example 10.2 64

AUTOMATING DOCUMENT BUILDING 66

Xtags Simple Batch Facility66

Using Scripts to Automate Document Building66

Get Text with Xtags 67
 AppleScript (Mac-only)..... 67
 ExtendScript (InDesign only)..... 67
 VB Script (InDesign/Windows only) 67
 Save Text with Xtags 67
 AppleScript (Mac-only)..... 67
 AppleScript (Mac-only)..... 67
 VB Script (InDesign/Windows only) 67
Example 11.1 68
 AppleScript (Mac-only)..... 68

ExtendScript (InDesign only).....	68
VB Script (InDesign/Windows only)	68
AppleScript (Mac-only).....	68
ExtendScript (InDesign-only)	68
VB Script (InDesign/Windows only)	69
CREATING XCATALOG AND INCATALOG LINKS	70
Adding Xcatalog/InCatalog Links to Anchored Boxes	71
APPLICATION-SPECIFIC TAGS.....	72
QuarkXPress-specific Tags.....	72
InDesign-specific Tags.....	72
<i>Example 13.1</i>	72
XTAGS SUMMARY	73
ERROR HANDLING	79
Parameters Out of Range.....	79
Error Alerts.....	79
Error Reports.....	79
AppleEvent Errors (MacOS only)	83

Chapter 1

Beginning with Xtags

What is Xtags?

Xtags™ is a text filter based on the XPress Tags language, but greatly enhanced, providing serious document-building power. Using Xtags, you import text files which it converts into fully formatted layouts, and you can export as well. Xtags does this by interpreting special formatting codes embedded in the text file, and we call this set of codes the Xtags language. Xtags adds several major features to the basic XPress Tags language, including the ability to:

- ◆ create and fill both anchored text and picture boxes, including shrinking boxes to fit their contents;
- ◆ create and fill both text and picture boxes outside of a text flow, including shrinking boxes to fit their contents;
- ◆ use relative values in tags;
- ◆ group created text and/or picture boxes;
- ◆ apply master pages to the current page or spread;
- ◆ copy and paste Xtagged-text between documents, or between applications;
- ◆ translate user-defined tags into standard Xtags constructs (or into anything else);
- ◆ perform table-based translations;
- ◆ use macros to express complex, parameterized tag sequences as shorthand (for example, fractions);
- ◆ provide advanced error reporting.

Xtags for QuarkXPress is also available in a “Pro” version. Xtags Pro adds a suite of table tags which enable advanced users to build sophisticated tables. Table tags are covered in chapter 8.

Except as noted in this manual, all XPress Tags constructs are supported by Xtags. Please see your QuarkXPress documentation for a detailed description of XPress Tags.

Xtags for InDesign, introduced in the first quarter of 2005, was designed with a syntax almost identical to Xtags for QuarkXPress. By doing this, we have ensured that codes written for one DTP application will port to another with a minimum of effort. Native InDesign tags are also indirectly supported (see page 72 for details).

The feature sets of QuarkXPress and InDesign are similar, but both applications have unique features not found in the other. Some differences in terminology or implementation are unavoidable. For example, QuarkXPress refers to text containers as text boxes; InDesign refers to them as text frames. QuarkXPress has a superior text style; InDesign does not, so the superior character style is mapped to InDesign's superscript style. These types of differences will be noted in the documentation as appropriate, like this:

InDesign Caveat: Unimplemented tag parameters are simply (and silently) ignored.

About This Manual

The main part of the manual describes the Xtags language, including most “vanilla” XPress Tags constructs. It is divided by chapter, where each chapter covers a group of tags of similar scope and purpose.

- ◆ *Appendix A* contains a listing of all Xtags tags and a brief definition of each tag.
- ◆ *Appendix B* contains a detailed description of Xtags error codes and error handling.

Before Installing Xtags. . .

What you should know about your computer

You should be familiar with basic Mac OS X or Windows concepts and procedures, such as using the mouse, selecting items from menus, entering information in dialogs, navigating among folders, and manipulating files (e.g. copying, renaming, and deleting).

What you should know about QuarkXPress/InDesign

Xtags will run on any computer that runs QuarkXPress 6, QuarkXPress 7, InDesign CS2, or InDesign CS3. This manual assumes that you are already comfortable with performing basic DTP tasks. You (or the person responsible for the document) should know how to:

- ◆ specify text formats: font, size, style, and so on;

- ◆ set paragraph formats, including indents, before and after spacing, tabs and rules;
- ◆ work with text and picture boxes;
- ◆ use the rulers, column guides and margin guides;
- ◆ edit text inside QuarkXPress/InDesign;
- ◆ define, apply, modify and delete paragraph and character style sheets;
- ◆ import text and graphics;
- ◆ set up and assign master pages;
- ◆ save and print publications.

It's not mandatory that you know these things, but it will be very helpful in shortening the learning curve.

Platform-specific issues

Xtags has been designed to work the same, or as closely as possible, in both applications and both operating systems. Differences between versions and applications will be noted as appropriate.

We will consistently refer to directories and subdirectories as “folders,” a term which is used in both Mac OS X and Windows parlance.

Note that Xtags itself fully supports Unicode, but it only usable on supported platforms (QuarkXPress 7.x and InDesign). Unicode can be used not only for the text flow, but also in translation files and macros, color names, style names, language names, line names, frame names, H&J names, box and layer names, etc.

Please download the latest version

Xtags is only available as a download from our web site; we do not ship physical copies of software. We regularly release maintenance updates to all our software, so please check our site regularly to make sure you are working with the most current release. The most recent public release can be found on Em's web site at

<http://www.emsoftware.com/products/xtags/download> .

and the most recent beta release (if any) can be found at

<http://www.emsoftware.com/products/xtags/beta> .

When you visit our download page, you will have four download choices:

- ◆ Xtags for QuarkXPress, Mac OS X version
- ◆ Xtags for QuarkXPress, Windows version

- ◆ Xtags for Adobe InDesign, Mac OS X version
- ◆ Xtags for Adobe InDesign, Windows version

Choose the right product for your situation, and download the appropriate zip archive. Each archive contains two folders, one for each supported version of QuarkXPress or InDesign. Each of these folders contains either the appropriate XTension (for QuarkXPress) or the appropriate two plug-ins (for InDesign)—the main plug-in plus InFlow. You should open the top-level folder corresponding to your version of QuarkXPress or InDesign, and install as described below.

Understanding Xtags version numbers

We know that in real-world use, many people who use our software do not use the most current shipping version of their page layout applications. We have always tried to maintain backward compatibility with at least one previous version release to accommodate this fact. That's why there are two versions of Xtags in every package we offer for download.

Our version numbers are tied to the host application's version number. Currently, for QuarkXPress, we have a 6.x version and a 7.x version, corresponding to versions 6.x and 7.x of QuarkXPress. For InDesign, we have a 4.x and a 5.x version, corresponding to InDesign 4 (called “CS2” by Adobe) and InDesign 5 (“CS3”). Our minor version numbers increase independently of the host application's as we release new features and fixes.

The feature sets of both supported versions are very similar, differing only where the applications are missing particular features (like opacity in QuarkXPress 6.x, for instance). As new versions of QuarkXPress and InDesign are released, we will continue our tradition of maintaining support for the most current version and the previous version.

Currently, we fully support Xtags 6.3 under QuarkXPress 6.x, Xtags 7.3 under QuarkXPress 7.x, Xtags for InDesign 4.1 under InDesign CS2, Xtags for InDesign 5.1 under InDesign CS3. We do not support Xtags 4.2, Xtags 3.x, or any version of QuarkXPress prior to 6.x. We do not support Xtags for any version of InDesign prior to CS2. Older versions of these XTensions and plugins are available for download, but we can't offer much support for them. They work the same as they always did (except lacking features of later products), and we can try to help you if you need it, but we can't fix bugs in them, or run the older products to help figure out what's going wrong, etc.

Installing Xtags

For QuarkXPress

With the appropriate folder open (see downloading information, above), copy the Xtags XTension file to the QuarkXPress application folder's XTension subfolder. If QuarkXPress is running, you will need to quit and relaunch it in order to make Xtags available.

Note that if you are using both QuarkXPress 6.x and 7.x, you will need to copy the appropriate version of Xtags to the XTension subfolder for each copy of QuarkXPress.

For InDesign

With the appropriate folder open (see downloading information, above), copy the Xtags plugin file to the Plugins folder within the InDesign application folder. If InDesign is running, you will need to quit and relaunch it in order to make Xtags available.

TIP: To make managing third-party plugins easier, you may want to create a subfolder in your Plugins folder and name it Em Software (or similar), and then put Xtags in that folder.

You will also need to install InFlow, a plugin product we provide that enables InDesign to automatically add pages as needed, like QuarkXPress's auto page creation feature.

Installing InFlow

When you purchase Xtags for InDesign, you also get an additional plugin we created called InFlow. InFlow adds pages automatically to your document as you import text and edit text in stories, just as you'd expect InDesign to do itself, and just the way that QuarkXPress has always done it. This means no more manually-created pages and text frames that you need to link by hand. InFlow is very easy to use, works in the background, and helps make InDesign behave like a well-adjusted citizen of the DTP community.

InFlow is useful even if you're not using it in conjunction with Xtags for InDesign. When you're ready to import a text document, you no longer need to hold down special keys to activate InDesign's specialized autoflow capabilities when placing text—just click the loaded cursor and let InFlow do the work.

InFlow works independently of any other plug-in, and can automatically flow placed text, typed text, or text generated by other plug-ins (such as our own InData and InCatalog, for which it was created).

Installation is the same as installing Xtags for InDesign. Detailed instructions on installation and use of InFlow can be found in the InFlow documentation. If you've misplaced your copy, you can download it from Em's web site at

<http://www.emsoftware.com/products/inflow/download> .

and the most recent beta release (if any) can be found at

<http://www.emsoftware.com/products/inflow/beta> .

Un-installing Xtags

For QuarkXPress

You can un-install Xtags by moving it out of the Xtensions folder or, just by deleting it. Moving an XTension to the XTensions Disabled folder (either manually or via the QuarkXPress XTension Manager) does not deactivate it from a licensing point of view. We recommend creating an XTensions Really Disabled subfolder within the QuarkXPress folder for such purposes. This is especially applicable for multi-pack versions of Xtags, where you need to run Xtags on another machine. In order to release its "slot" on the network, you must move the XTensions file to such a "really disabled" folder to make sure it's truly disabled.

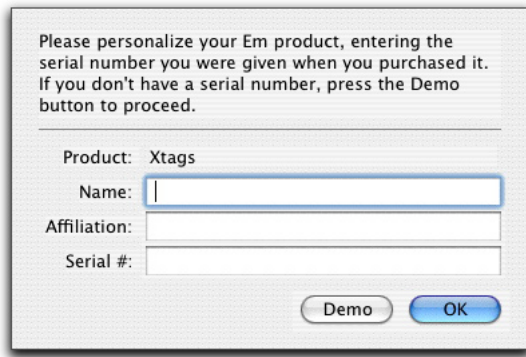
For InDesign

You can un-install Xtags by moving the Xtags plugin and the InFlow plugin entirely out of your InDesign/Plugins folder or InDesign/Plugins/Em Software subfolder, or just by deleting them.

TIP: To make managing third-party plugins easier, you may want to create a subfolder in your InDesign main folder and name it Plugins Disabled (or similar), and then put Xtags in that folder.

Personalizing Your Copy of Xtags

Your serial number is made available to you after you purchase the serial (either online or via email). Be sure to record your Xtags serial number somewhere safely and permanently. The first time that you start up QuarkXPress or InDesign after installing Xtags, you will be prompted to fill in your name, your organizational affiliation (this is optional), and your Xtags serial number:



QuarkXPress 6.x, Mac OS X

Once entered, the serial number can be changed (e.g. upgraded to increase the number of simultaneous copies, etc.) by selecting one of the following and pressing the **Upgrade...** button:

- ◆ (QuarkXPress) **Utilities**→**Xtags**→**About...**
- ◆ (InDesign CS2/Win) **Help**→**About Em Software Plug-ins**→**Xtags™ ...**
- ◆ (InDesign CS2/Mac) **InDesign**→**About Em Software Plug-ins**→**Xtags™ ...**
- ◆ (InDesign CS3/Win) **Help**→**About Plug-Ins**→**Em Software**→**Xtags™ ...**
- ◆ (InDesign CS3/Mac) **InDesign**→**About Plug-Ins**→**Em Software**→**Xtags™ ...**



Upgrading in InDesign CS3, Windows XP

If you need to do more serious testing than the demonstration version supports, please [contact sales](#) with a brief explanation of what you're trying to do, and we can issue a temporary serial number.

If you don't have a serial number, and just want to try the demonstration version of Xtags, press the **Demo** button to proceed from the serialization dialog. The only limitation in demonstration mode is that you can't import or export more than 50 paragraphs at a time, which should be adequate for fairly simple testing.

Chapter 2

Using Xtags Interactively

Conventions Used in the Documentation

The chapters that follow describe the tag language of Xtags. As we document the various tag functions, we're going to show you lots of examples of Xtags in use. We believe that the best way to understand Xtags constructs is in context, so the text used for these examples is intended to both explain and illustrate the tag's function. In each case, we present both the original source text containing Xtags codes and the formatted results you would get by importing the text into your document with Xtags. The source text sometimes includes explicit carriage return and tab characters (denoted by ¶ and →, respectively) to avoid ambiguities. To make them easier to spot, tags in the source text will be set in a different typeface, `<f"Courier"><f"Courier">like this</f$></f$>`.

So before you begin, please familiarize yourself with the following formats used for examples, Xtags tags, and comments. Here's a short example showing source Xtags and their results:

Example 2.1

Xtags Code:

```
@$:Here is an example of the <B>f<B> tag: <f"Zapf
Dingbats">u<f$> is a great bullet character.
```

Formatted Result:

Here is an example of the f tag: ◆ is a great bullet character.

If the first paragraph of an example begins with the @\$: tag, it means that paragraph is applying the **Normal** ([**Basic Paragraph**] in InDesign) style sheet to the text. In other examples, the style sheet **Example** is used. Both of these style sheets are very similar to the regular text paragraphs within this manual.

Xtags code examples will be shown in Courier, like this:

```
<*L>This is a left-justified paragraph.¶
```

Note that in the code examples, we explicitly state line and paragraph endings. When we need to talk about tag elements within a paragraph, we'll again set them in boldface Courier. For example, the previous example illustrates the `<*L>` tag.

When we are discussing tag syntax, we will use italic Courier type to indicate general elements (i.e., parameters within tags) which will need to be replaced by specific text when used in a tag input sequence, as in this example:

```
<*L>This is a left-justified <f"font-name">paragraph.¶
```

Here, *font-name* is a placeholder for a font name that you would provide when you actually use this tag.

When we add comments to tags, we will set them in regular italic type to distinguish them from the tag itself, as in this example:

```
<f"Times"> Set font to Times
```

The names of the application menus items, Xtags menu items, and dialog items are set in boldface type. For example, "Select **Open...** from the **File** menu," and "Click the **Cancel** button to change your mind."

We use arrows (→) to represent paths through a series of nested menus: the form **Utilities**→**Xtags** means that you should select the **Utilities** menu, and then select **Xtags** from the **Utilities** menu.

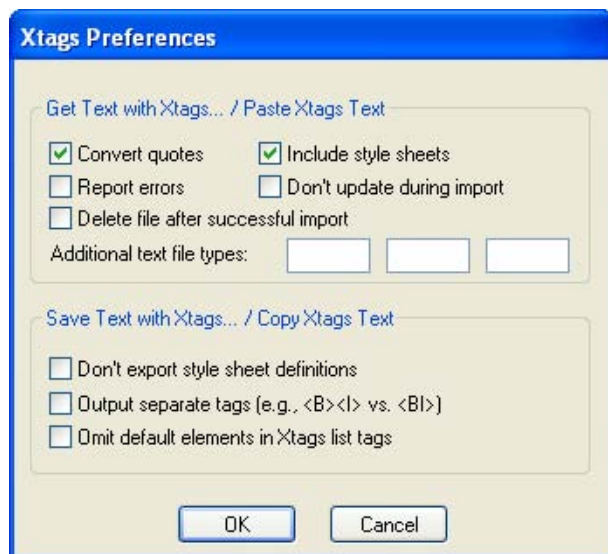
Finally, file and folder names within regular text are set in plain Helvetica type, as in "this file should be placed in the XTensions folder."

Xtags Preferences

In QuarkXPress, Xtags preferences are located under **Utilities**→**Xtags**, and in the general preferences (**QuarkXPress**→**Preferences**→**Xtags**).

In InDesign, Xtags preferences are found under **InDesign**→**Preferences**→**Xtags**.

The **Xtags Preferences** dialog allows you to specify a variety of Xtags defaults. Note that these default settings persist across application restarts. It is divided into two areas, controlling importing text (**Get Text with Xtags**) and exporting text (**Save Text with Xtags**):



QuarkXPress 7.x, Windows XP

The **Get Text with Xtags.../Paste Xtags Text** area sets options for Xtags import operations. The five check boxes have the same meanings as those in the **Get Xtags Text** dialog.

Convert quotes

tells Xtags to automatically change any straight quote (" or ') to the appropriate printer's quote (including various European quote pairs), depending on its context and the current document's "smart quotes" preferences: "sample" becomes "sample" and 'sample' becomes 'sample'.

Include style sheets

tells Xtags to interpret tags as formatting instructions. This must be checked for normal Xtags operation.

Report errors

tells Xtags to insert a brief error message into the text being built, if it encounters any problems while importing the file, and to alert you to the total number of errors when it's done. Xtags continues to process text even when it finds an error, so if you don't select this option, Xtags will do its best in the face of errors and never tell you a thing. The error messages themselves are inserted in-line at each point of error, in the form

```
«Xtags error: description: tag info»
```

where *description* is a textual description of the error, and *tag info* specifies information about the tag and parameter that triggered the error. These error

messages and their meanings are documented in Appendix B, "Error Handling."

Don't update during import

tells Xtags to suppress document window updating during importing. Selecting this option will usually speed up importing, particularly when the imported text contains many anchored boxes.

Delete file after successful import

tells Xtags to delete the imported file if the import operation is successful. Note that the file is actually deleted, not just moved to the trash.

The **Additional text file types** fields allow you to specify up to three additional file types which correspond to text files on your system. File types consist of four characters under Mac OS X and are designated by file extensions of one to three (usually three) characters under Windows.

The **Save Text with Xtags.../Copy Xtags Text** area sets options for Xtags copy and save (export) operations.

The check boxes in this area have the following meanings (which are generally self-explanatory):

Don't export style sheet definitions

tells Xtags to exclude tags defining style sheets from the saved or copied Xtags text.

Output separate tags

By default, all combinable tags are merged into a single tag in the exported text. For example, the tag for bold italic text is output as `<BI>`. When this option is checked, then tags are always output individually; in our example, bold italic text would be tagged as `<I>`.

Omit default elements in Xtags list tags

When this option is checked, then elements in tags which have the default value for that tag are omitted (left empty) from the output text. By default, all elements are filled-in in the output tags. For example, these tags illustrates the output format for a 6-point 50% (100% opacity) blue rule above a paragraph in the default output mode:

```
KEEP DEFAULT ELEMENTS
```

```
<*ra(6,0,"Blue",50[,100],0,0,0)>
```

and these tags are what you get for the same thing, but omitting default elements:

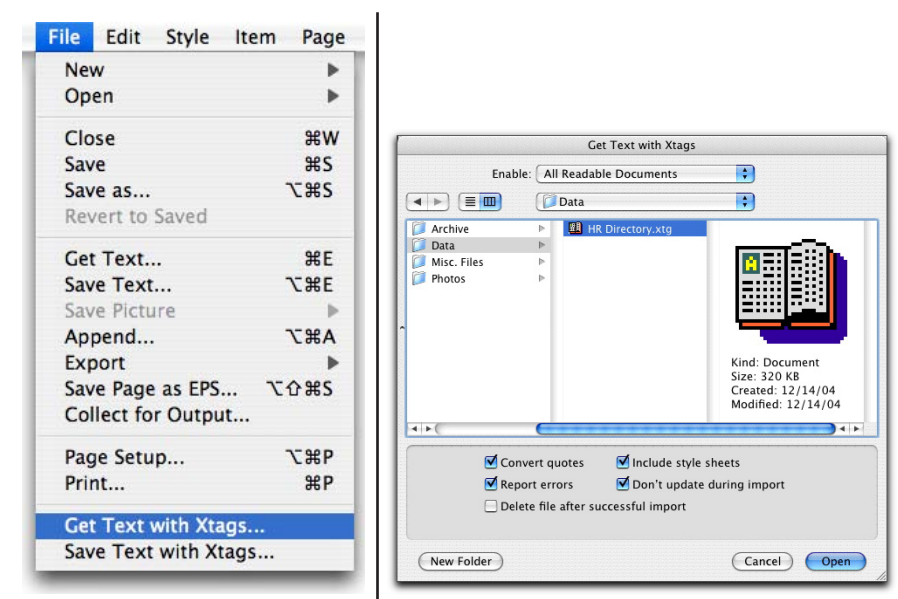
```
OMIT DEFAULT ELEMENTS
```

```
<*ra(6,, "Blue", 50[, ], , , )>
```

Note: The parameter within brackets indicates the rule opacity, and only exists only for tags marked as `<v7.00>` or later (see page 23).

Importing Text With Xtags

To import a text file using Xtags, make sure you're in content mode, with a selection (empty or not), and choose **File**→**Get Text with Xtags...** (**File**→**Import Text with Xtags...** in QuarkXPress 7.x). This opens the **Get Text with Xtags** dialog:

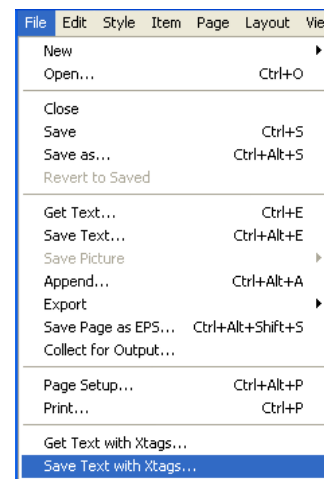


QuarkXPress 6.x, Mac OS X

The settings at the bottom of the dialog are initially determined by your Xtags preferences (see *Xtags Preferences* earlier in this chapter). You can override those settings each time you get text with Xtags, or you can change it permanently by changing them in the Xtags preferences. When the settings are set the way you need them, choose the appropriate file to begin the import process.

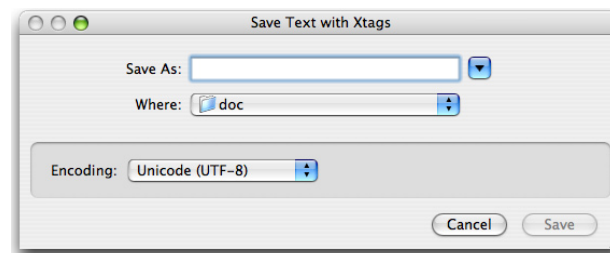
Saving Text with Xtags

The **Save Text with Xtags...** menu item in the **File** menu may be used to save text styled with Xtags (suitable for subsequent importing):



QuarkXPress 7.x, Windows XP

This option saves any selected text or selected item(s) to the file you specify. The default file name is the name of the current document with the extension `.xtg` added. Options set in the **Xtags Preferences** dialog are in effect for these save operations. The desired output encoding may be chosen from the dialog's drop-down menu.



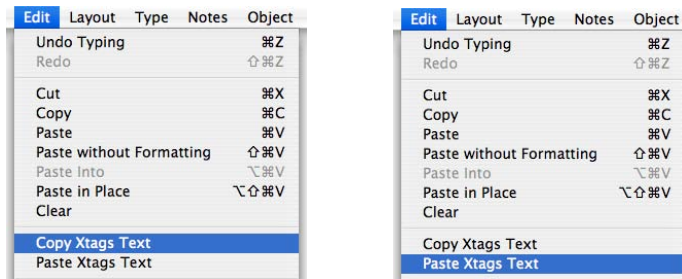
InDesign CS3, OS X

Characters that cannot be represented in the target encoding will be converted to `<\#U+xxxx>` tags. When saving/copying to any Unicode encoding, we don't convert any code points to tags.

- ◆ If no text is selected when you choose **File**→**Save Text with Xtags...**, the entire current story will be saved.
- ◆ If an unanchored box or group is selected, then the tags necessary to recreate that unanchored box or group will be saved.
- ◆ Multiple items may be selected; their corresponding tags will be generated in the order required to re-create the appropriate back-to-front order.
- ◆ Any anchored boxes included among the saved text will include the tags required to recreate that anchored box in the corresponding location in the exported text.

Copying and Pasting Xtags Text

Xtags also adds two items to the **Edit** menu, as follows.



InDesign CS3, Mac OS X

Copy Xtags Text (Copy with Xtags in QuarkXPress 7.x)

Turns the selected text, picture box or group into Xtags input on the system clipboard, suitable for pasting normally or with **Edit**→**Paste Xtags Text**.

Paste Xtags Text (Paste with Xtags in QuarkXPress 7.x)

Import text on the system clipboard, interpreting it as Xtags input.

When you **Copy Xtags Text**, the save/export settings in the **Xtags Preferences** dialog are used for this operation. When you **Copy Xtags Text** and it's not just a text selection being copied (i.e., if you're copying a box or group), Xtags will create a graphic preview (a BMP under Windows, a PICT under Mac OS) that you can paste in another application (like Photoshop or FileMaker) or into a graphic box in XPress or InDesign itself.

The **Copy Xtags Text** action can be very useful for learning which tags create an effect you want to achieve. Simply style some sample text, a sample picture box, or grouped items; select it, choose **Edit**→**Copy Xtags Text**, and then paste it as

normal text back into some convenient location (perhaps a text box on the pasteboard) with the normal **Edit**→**Paste** command.

Paste Xtags Text is enabled only when there is text present on the system clipboard and the cursor is positioned within a text frame. (This is required even for pasting unanchored objects.) The import settings in the **Xtags Preferences** dialog are used for this operation.

The **Paste Xtags Text** action may be used to quickly preview the results of some tagged text you are creating. Select the raw text you want to preview and copy it to the system clipboard, place the insertion point in some convenient target location (perhaps a text box on the pasteboard), and choose **Edit**→**Paste Xtags Text**. This will create the the formatted text and associated boxes.

Chapter 3

Learning Xtags Basics

General Information

Xtags is a tagging or “mark-up” language, a family of formatting codes design to be inserted in a text file, allowing you to build documents hands-off. These codes are placed within angle brackets: for example, `<z10>` sets the font size to 10 points. Multiple codes may be concatenated within one set of brackets: e.g., `<BIz10cK>` is equivalent to `<I><z10><cK>` (specifies bold italic ten-point black type).

Unlike other tagged text languages like HTML, all tags are *case-sensitive* in general (though their parameters generally are not). For example, `` and `` mean entirely different things.

Xtags source files are formatted as a sequence of paragraphs, each containing regular text and, optionally, tags: any character not inside a tag sequence (`<...>`) is input normally. Note that all spaces and tabs outside of tags are significant. Paragraphs are ended by an end-of-line character or sequence (carriage return and/or line feed), or by a column/page break character of some kind.

Character formatting tags—such as point size and font specifications—take effect immediately, and last until explicitly or implicitly changed. Paragraph formatting tags—such as justification and indent settings—may be placed anywhere in the current paragraph (but before the terminating paragraph return, new column, or new box character), take effect immediately over the whole paragraph, and last until explicitly or implicitly changed. (Implicit changes usually occur when you apply a named style to a new paragraph.) If there are multiple, conflicting paragraph formats applied to a single paragraph, the last format applied “wins.” Example 3.1 shows this in action.

Example 3.1

Xtags Code:

```
<*L>This is a left-justified, ah, a <*C>centered
paragraph.¶
```

Formatted Result:

This is a left-justified, ah, a centered paragraph.

The paragraph in the example will be centered since the `<*C>` (center) tag occurs after the `<*L>` (left-justify) tag in the same paragraph.

Control characters

Common control characters are mapped to these characters:

CODE	CORRESPONDING CHARACTER
7	shift-return (hard return, i.e., a line break)
9	tab
11	vertical tab (new column)
13	carriage return (new paragraph)
15	punctuation space
16	flexible space
29	discretionary return
30	temporary margin
31	discretionary hyphen

InDesign Caveat: Code 29 (discretionary return) is mapped to a zero space break.

Word spaces within tags

Word spaces may be placed before and after any *complete element* (tag or parameter). For example, the following tags are legal in Xtags:

```
< z 10 > and < *p(1", 0, 1", 12, 0, 0, g) >
```

while the following tags are illegal in Xtags:

```
<z1 0> and <* p(1 ", 0, 1 ", 12, 0, 0, g)>
```

These tags are illegal because the space occurs within a number or measurement value (`1 0` and `1 "`) or within a multi-character tag (`* p`).

Splitting long tag sequences

Xtags lets you to split a long sequence of tags over two or more lines in the text file by placing a colon at the end of each line¹ to be continued (separating lines with carriage returns).

Here is an example which defines a style named **Indented**, based on the Normal style:

```
@Indented=[S"Normal"]<*p(1",0,1",12) :¶
f"New Century Schoolbook"z10*L*h"Medium">¶
```

Using default settings

You can use a dollar sign (\$) code in place of most XPress Tag parameters, standing for the default value specified in the current style sheet. If there is no current style sheet—i.e., if the current paragraph's style sheet displays as **No Style** when you look under **Style Sheets** on the **Style** menu—then the attribute takes on the default value in the **Normal** style sheet. This is the best way to reset a value once you're done with some special setting.

InDesign Caveat: The **Normal** paragraph style, which is predefined in QuarkXPress, will be mapped to InDesign's default [**Basic Paragraph**], but only if there's not another paragraph style named **Normal** in the document. Similarly, if there is no character style named **Normal**, Xtags will map it to InDesign's [**None**] entry.

Tag Parameters—The Heart of Xtags

Tag parameters are sub-lists of additional controls for a particular tag. Let's say you want to create a text box—that's the **&tbu2** tag. But this isn't very helpful—you need to know what size the box is, where to put it, and what to fill it with. All that additional information is contained in parameter tags.

When we're talking tag parameters, tags come in four flavors.

1. *The basic tag.* No additional parameters are need. Examples of this would be character attributes—**** (bold) or **<K>** (all caps).
2. *Single parameter tags.* Some tags, such as the font size tag (**<f"font name">**), require a single parameter following the tag.
3. *Multi-parameter tags.* Some tags, such as the drop caps tag (**<*d(chars, lines)>**), which we'll cover in chapter 5, have multiple parameters.

4. *Multi-parameter tags with complex sub-parameters.* Some tags, such as the text box tag (which we'll cover in chapter 7), have multiple parameters, and some of those parameters can be further specified as sub-lists.

Tags with single parameters

Some tags require a single parameter following the tag. If the parameter is a text string, it must be enclosed in double quotation marks (either straight double quotes or matching printer's double quotes). For example, **<f"Times">** sets the font to Times. Literal quotation marks and other special characters may be included within a string by preceding the desired character with a backslash character.

When a tag requires a measurement parameter, the number is assumed to be in points by default. Xtags also allows you to specify other measurement systems by including the appropriate units. For example, both of these tags set a left tab stop with (no leader character) at 2 inches:

```
<*t(144,1," ")> or <*t(2",1," ")>
```

Note that there's no confusion with the double quote mark that's part of the 2" value, because it doesn't begin the parameter value, unlike the space in double quotes (the third parameter).

For tags that take only a single numeric parameter (and hence don't normally require parentheses), you can specify the parameter in units other than points by placing the parameter in parentheses. For example, **<z(1p2)>** sets the font size to 1 pica 2 points, or 14 points. (For QuarkXPress, the XPress Tags equivalent is **<z14>**.)

Xtags also supports relative values for numeric arguments in those tags where it makes sense: **z** (size), **s** (shading), **h** (horizontal scaling), **y** (vertical scaling), **t** (tracking), **b** (baseline shift), **br** (relative baseline shift), and ***p** (paragraph settings). When a parenthesized parameter to these tags is started with a period followed by an arithmetic operator (+, −, * or /), the value is interpreted mathematically. For example:

- ◆ The tag **<z(.+2)>** tells Xtags to increase the current font size by two points (default units).
- ◆ The tag **<s(.−20)>** says to decrease the current shading by 20% (default units, but of a different sort).
- ◆ The tag **<z(.−p3)>** or **<z(.−3pt)>** tells Xtags to decrease the current text size by 3 points (explicit units).

¹ The colon must be placed between tags, not in the middle of a tag.

When a tag is preceded by a period and the multiplication or division sign, the current value for that setting is multiplied or divided by the specified parameter value. For example, the tag `<h(. *2.5)>` sets the horizontal scaling to two and a half times its current value. Note that the number following `*` and `/` must be unitless — `<h(. *2.5p)>` would generate an error.

Tags with multiple parameters

Some tags require multiple parameters following the tag. For example:

- ◆ The drop caps tag: `<*d(chars, lines)>`
- ◆ The rule above tag: `<*ra(thickness, style, color, shade, [opacity,]2 from left, from right, offset)>`

Everything that applies to a single parameter tag applies to a multiple parameter tag, with the following addition: Multiple parameters are always enclosed in parentheses and separated by commas. Spaces between the comma and the parameter (as seen above) are optional; we've included them in all tag definitions to make it easier to read. When you save Xtags text as output, parameters will not have spaces between them.

Tags with multiple parameters and sub-list parameters

Some tags are extremely complex. To simplify the specification of complex parameter cases, Xtags supports a “sub-list” parameter specification scheme. This construct allows you to specify another list inside a list, with the inner list being treated syntactically as a single parameter. We use the sub-list scheme to expand the parameter options for some tags without introducing new top-level list parameters that would force us to go to a whole new tag scheme, and to let us expand the options available for a single parameter without inventing complex sub-languages for parameters that sometimes involve more than one piece of information.

The text box tag is a good example of this:

```
<&tbu2((0, TL, 2), 2, 72, 144)>...<&te>
```

The `&tbu2` tag creates a text box, and it expects the box's x and y coordinates and width and height as its first four parameters. In this case, however, we have replaced the first parameter, x, with a sub-list `(0, TL, 2)`. This sub-list translates to `0` for the *x offset*, `TL` for “top left,” and `2` for the relative box reference. In plainer language, it means that the box's reference x coordinate should be offset 0 points in the horizontal dimension (and 2 in the vertical dimension,

from the second argument above) from the top left of the second-most-recently-created box.

Omitted parameters

Unlike XPress Tags, Xtags allows you to omit unneeded parameters from multiple-parameter tags. For most tags, Xtags will set omitted parameters to the default value. For example, if you omit the color parameter in the `*ra` (rule above) tag, then it will default to **Black**, which is the default value for rules in QuarkXPress.

Just like list elements, sub-list elements may be omitted to use their default value. For example, the three-element sub-list `(, t1,)` or just `(, t1)` omits the first and third parameters, using whatever default values they might have. Finally, the sub-list `(1)` (a single element) behaves exactly like a simple `1` (the same single element).

Parameters omitted from the paragraph settings tag (`*p`) are handled differently by Xtags. Here, an omitted parameter has the effect of leaving the corresponding setting unchanged. For example, the tag `<*p(1", , 1")>` changes both margins to one inch, and leaves all other paragraph settings unchanged. This example also illustrates the relevant syntax: *parameters omitted between specified parameters require a comma to keep their place, while those coming after the last specified value can simply be omitted.* (The above example could also have been specified as `<*p(1", , 1", , , ,)>`.)

As in XPress Tags, any tag with a single numeric parameter may omit a zero parameter. Thus, `` is equivalent to `<b0>` (set the baseline shift to zero), but this is generally considered “bad form”.

Example 3.2

This example is designed to illustrate the features and power of Xtags. We've included comments to explain what the various codes do. The first 4 lines of code define paragraph styles.

Xtags Code:

This is a style sheet definition, named “Ad Head.” It uses 12 point bold Helvetica type (with 13 point leading), all caps, and is centered. It has a 14 point black rule and white text, making this a reverse head. All margin settings are 0, and the space after and space before are set to 12 points. All lines remain together at both the beginning and the end of the paragraph, and it will “keep with next”:

```
@Ad Head=[S" , "Ad Head"]<*C><*h"Standard"><*kn1><*kt(2,2)>
<*ra(14, "Solid",K,100,100†,0,0,-2.52)><*rb0><*d0><*p(0,0,0,
13,12,12,g, "U.S. English")><B><H><s100><t0><h100><z12><k0>
<b0><cW><f"Helvetica">¶
```

2 The opacity parameter only exists in tags marked as `<v7.00>` or later

This paragraph style is named "Ad Regular." It uses 10 point plain Helvetica type (with 11 point leading) and is justified. All margin settings are 0, and the space after setting is 12 points. Two lines must remain together at both the beginning and the end of the paragraph:

```
@Ad Regular=[S"" ,"Ad Regular"]<*J><*h"Standard"><*kn0><*kt
(2,2)><*ra0><*rb0><*d0><*p(0,0,0,11,0,12,g,"U.S. English")
><P><s100><t0><h100><z10><k0><b0><cK><f"Helvetica">¶
```

This style sheet is similar to "Ad Regular, but adds 50% black rules above and below the paragraph and increases the space before to 6 points:

```
@Ad with Rules=[S"" ,"Ad with Rules"]<*J><*h"Standard"><*kn
0><*kt(2,2)><*ra(2,"Solid",K,50,100†,0,0,20%)><*rb(2,"Solid
",K,50,100†,0,0,10.001%)><*d0><*p(0,0,0,11,6,12,g,"U.S. Eng
lish")><P><s100><t0><h100><z10><k0><b0><cK><f"Helvetica">¶
```

This style sheet is similar to "Ad Regular," but it includes an anchored graphic:

```
@Ad with Pix=[S"" ,"Ad with Pix"]<*J><*h"Standard"><*kn0><*
kt(2,2)><*ra0><*rb0><*d0><*p(0,0,0,11,0,12,g,"U.S. English
")><P><s100><t0><h100><z10><k0><b0><cK><f"Helvetica">¶
```

All the code at the front of this defines the text box and its location. Then we have the first line of text, in this case a heading:

```
<&tbu2(408,81,360,437.083,,,,,(,n),(,100),,
n,,(1,1,1,1),2,12,,,,,)>@Ad Head:apartments for rent¶
@Ad Regular:Large 2-bedroom apartment in a quaint old
Victorian house. Close to subway stop. Perfect for a young
couple or family. $550/all utilities included. Call Mrs.
Jones, 555-2122.¶
```

Notice that this paragraph did not begin with a style name. If a style name is omitted, it inherits the style from the previous paragraph.

```
Several 1-bedroom apartments near the University in a
popular student building. Enjoy our quiet yet social
atmosphere. $245/utilities extra. Hulbert Arms Apartments,
Manager, 555-3409.¶
```

```
@Ad with Rules:Spacious 2-bedroom apartment in Chelsea-
Bonnevillle area in a quiet family neighborhood. All wood
floors and a master bedroom suite with a walk-in closet.
Parking available for one car as well as basement laundry
hookups and storage space. A real gem, won't last long.
$875/all utilities included. Contact Mr. Riverside,
444-2211.¶
```

```
@Ad Regular:Basement apartment available, perfect for
student/<\d>recent graduate. Large 1-bedroom flat with
working fireplace and many amenities. $445/heat included.
James Wills, 443-9865.¶
```

```
In-law apartment available in our home for non-smoker
only. $450/heat included; rent reduction is possible in
exchange for child care or light yard work. Call J. Pierce,
555-4435, evenings only.<\c:¶ This is the "new column" code
```

```
>@Ad Head:houses for sale¶
```

```
@Ad Regular:Mint-condition 3-bedroom house in central
Chelsea now available. Quiet family neighborhood is
perfect for a growing family. Elegant master bedroom suite
has a large walk-in closet. 1 car attached garage and a
wonderful backyard garden are just two of the extras that
come with this gem. Priced to sell at $123,995. Call Tim
Spin to view this fine home, 443-8854.¶
```

All the codes immediately after the style sheet name are related to the anchored photo. Within the ad, local character formatting tags create the fraction and the italic phrase.

```
@Ad with Pix:<&pb(102.519,61.511,a,0.5,(,),(,100),,,,(,
4,6),m,20,20,-22.826,-21.468,,,):Farm and outbuildings.
jpg",,)><B>A Treasure!<B> Magnificent property available
in Willbury Valley. 10 room farmhouse on 50 acres, built
in 1875, fully updated. Glorious gourmet kitchen, 4
bedrooms, 3<z7><k-30><b3>1<z10><k0><b0>/<z7>2 <z10>baths.
Includes a finished basement with a guest room. Includes 4
outbuildings and a 2 acre pond. <I>At $2.6M, this property
will not last long.<I> Call Janice Snowden to arrange a
private showing of this exceptional home, 555-0985, X321.¶
@Ad with Rules:Duplex available in Wooster Square area.
Perfect for a large family, an in-law arrangement, or as
rental property. All new plumbing and electrical installed
within the last five years. $435,000. Call Tim Spin for an
appointment to see this great property, 443-8854.
```

Note: The opacity parameter is marked with †. This parameter should be omitted except when importing in <v7.00> mode (see page 23).

Formatted Result:

APARTMENTS FOR RENT

Large 2-bedroom apartment in a quaint old Victorian house. Close to subway stop. Perfect for a young couple or family. \$550/all utilities included. Call Mrs. Jones, 555-2122.

Several 1-bedroom apartments near the University in a popular student building. Enjoy our quiet yet social atmosphere. \$245/utilities extra. Hulbert Arms Apartments, Manager, 555-3409.

Spacious 2-bedroom apartment in Chelsea-Bonneville area in a quiet family neighborhood. All wood floors and a master bedroom suite with a walk-in closet. Parking available for one car as well as basement laundry hookups and storage space. A real gem, won't last long. \$875/all utilities included. Contact Mr. Riverside, 444-2211.

Basement apartment available, perfect for student/recent graduate. Large 1-bedroom flat with working fireplace and many amenities. \$445/heat included. James Wills, 443-9865.

In-law apartment available in our home for non-smoker only. \$450/heat included; rent reduction is possible in exchange for child care or light yard work. Call J. Pierce, 555-4435, evenings only.

HOUSES FOR SALE

Mint-condition 3-bedroom house in central Chelsea now available. Quiet family neighborhood is perfect for a growing family. Elegant master bedroom suite has a large walk-in closet. 1 car attached garage and a wonderful backyard garden are just two of the extras that come with this gem. Priced to sell at \$123,995. Call Tim Spin to view this fine home, 443-8854.



A Treasure! Magnificent property available in Willbury Valley. 10 room

farmhouse on 50 acres, built in 1875, fully updated. Glorious gourmet kitchen, 4 bedrooms, 3½ baths. Includes a finished basement with a guest room. Includes 4 outbuildings and a 2 acre pond. At \$2.6M, this property will not last long. Call Janice Snowden to arrange a private showing of this exceptional home, 555-0985, X321.

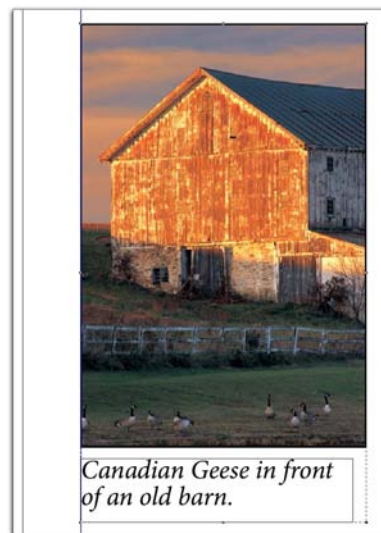
Duplex available in Wooster Square area. Perfect for a large family, an in-law arrangement, or as rental property. All new plumbing and electrical installed within the last five years. \$435,000. Call Tim Spin for an appointment to see this great property, 443-8854.

Tips for Constructing and Debugging Tags

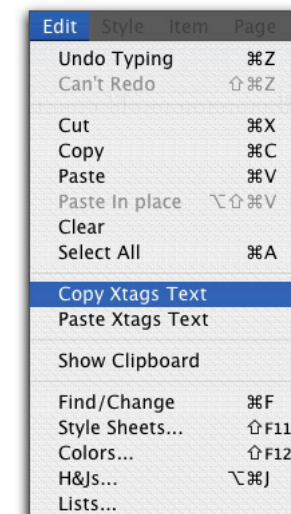
The **Copy Xtags Text** facility can be very useful for learning which tags create an effect you want to achieve: simply style some sample text, a sample text or picture box, or a sample group as you desire, select it, choose **Edit→Copy Xtags Text** (**Edit→Copy with Xtags** in QuarkXPress 7.x), and then paste it back as normal

text into some convenient location (perhaps a text box on the pasteboard) with the normal QuarkXPress **Edit→Paste** command, or into another application with its own **Edit→Paste** command.

Example 3.3



When we grouped this photo and caption . . .



*. . . then copied the grouped items via **Copy with Xtags** . . .*

. . . we got this when we pasted the result:

```
<&o(50,129)><&pbu2(0,0,117.864,175.292,,,,,0.5,(,),(,100),,
,,(,4,6),m,27.176,27.232,,,,,"Old Barn.jpg",,,)><&tbu2(0,1
80.213,112.354,26.111,,,,,n,,(,n),(,100),,,,,,)@&:$:<*
L><&p(0,0,0,11,0,13.032,g,"U.S. English")><I>Canadian Geese in
front of an old barn.<&te><&g(2,1)>¶
```

Similarly, the **Paste Xtags Text** facility may be used to quickly preview the results of some tagged text you are creating. Select the raw text you want to preview and copy it to the system clipboard with the normal **Edit→Copy** command. Then, place the insertion point in some convenient target location (again, a text box on the pasteboard is often a good choice), and then choose **Edit→Paste Xtags Text** to preview the formatted text produced by the tag sequence.

Chapter 4

Setting Character Attributes

Reset Attributes Tags

- `<a$>` Reset all character attributes to the character attributes specified by the currently-applied paragraph style sheet.
- `<a$$>` Reset all character attributes to the character attributes specified by the currently-applied character style sheet.

Character Face Tags

Character face tags are used to specify character treatment of text. These tags generally consist of a single character which toggles the current state of the various character face attributes. The tag code characters have the following meanings:

- | | | | |
|---------------------------|--|------------------------|-----------------------|
| <code><P></code> | Sets plain face | <code></></code> | Toggle strike-through |
| <code></code> | Toggle bold | <code><K></code> | Toggle all capitals |
| <code><I></code> | Toggle italic | <code><H></code> | Toggle small capitals |
| <code><O></code> | Toggle outline | <code><+></code> | Toggle superscript |
| <code><S></code> | Toggle shadow | <code><-></code> | Toggle subscript |
| <code><U></code> | Toggle underline | <code><V></code> | Toggle superior |
| <code><W></code> | Toggle word underline | | |
| <code><\$></code> | Set to current paragraph style's face | | |
| <code><\$\$></code> | Set to current character style's face. | | |

InDesign Caveat: The outline and shadow face tags (`<O>` and `<S>`) are ignored since there's no comparable character style in InDesign. The superior face tag does not exist in InDesign; we've mapped this tag to the superscript tag.

The `<P>` tag operates somewhat differently than the others in that it resets the character face to plain text rather than toggling any settings.

Generally, character face tags toggle settings. However, if you prefix a face-toggling tag with a tilde character (~), the tag becomes a face-setting tag. For

example, the tag `<~B~I>` always turns on both bold and italic faces, regardless of whether they are set already or not, whereas `<BI>` would simply toggle the current state of both bold and italic. Remember that `<P>` turns off all (non-plain) character face attributes.

InDesign Caveat: In QuarkXPress, applying the `I` and/or `B` tags to a given font automatically selects any linked fonts in the font family. Thus, applying `I` when the current font is Times will leave the font as Times, with the italic attribute; when QuarkXPress outputs the document, it will use the actual Times Italic font. This can be a bit tricky, as you can call in an italic format without an actual italic font, and you won't know you have problem until you output the file.

InDesign is a better prepress citizen in this regard—InDesign will immediately change the font from Time to Times Italic, if it's available. If it's not available, it will highlight the text (magenta is the preset color, but you can change that to whatever you like) to inform you that there's something wrong with the font being called in. So an attribute combination like `<f"Times"><I>` that imports successfully into QuarkXPress may result in a "font not found" error when importing the same tags into InDesign if the variant "Times Bold Italic" doesn't exist.

Example 4.1

Xtags Code:

@Example: Most character formatting tags toggle the current state of their setting. So, to make a word italic, simply surround it with the appropriate tag `<I>` like so `<I>`. The first tag turns italic on, and the second turns it off.¶ You can combine character formatting tags: `<BI>` this text will be bold italic `<P>`. Notice how the plain text tag turns all formatting off.¶

@Example Last: The dollar sign tag is special; it restores the paragraph's default settings. For example, in this paragraph, which has italic text defined in its style sheet, if we turn on boldface, `` we get bold italic type. Using the italic tag `<I>` will turn off italics in this case, leaving boldface type. `<$>` As you can see, using the default character format tag restores the typeface of the paragraph's current style sheet.¶

Formatted Result:

Most character formatting tags toggle the current state of their setting. So, to make a word italic, simply surround it with the appropriate tag *like so*. The first tag turns italic on, and the second turns it off.

You can combine character formatting tags: *this text will be bold italic*. Notice how the plain text tag turns all formatting off.

The dollar sign tag is special; it restores the paragraph's default settings. For example, in this paragraph, which has italic text defined in its style sheet, if we turn on boldface, we get bold italic type. Using the italic tag will turn off italics in this case, leaving boldface type. As you can see, using the default character format tag restores the typeface of the paragraph's current style sheet.

Character Size and Font Tags

The **z** and **f** tags set the type size and font, respectively:

<code><zsize></code>	Set size, in points
<code><f"font name"></code>	Set font, by "font name"

The **z** tag sets the point size to the specified (absolute or relative) size (minimum 2 pt; maximum 720 pt).

The **f** tag sets the font to *font name*. The given font name may be either a name found in the current document's font menu or the equivalent PostScript font name. Make sure to use correct capitalization in font names—`<f"times">` won't work.

Example 4.2

Xtags Code:

@Example:The font name and size tags are also very easy to use. The `f` tag changes the font:¶
`<f"Helvetica">`This paragraph is in Helvetica. (We had to spell out "Helvetica" completely<\m>don't forget to do that).
 We can also change the type size: `<z8>`from small to `<z18>`very large `<z$>`with the `z` tag. Of course, we'd probably also want to change the leading at the same time, but that tag is still to come...¶
 You can reset things back to normal, either by explicitly changing the setting back to its previous value (as we

did earlier), or with the `$` parameter, like so, `<f$>`which restores the font to that specified by the current paragraph's style sheet.¶

@Example Last:Xtags also lets you use a relative value for many parameters. Here, for example, `<z(.+2)>`we increase the current size by 2 points.¶

Formatted Result:

The font name and size tags are also very easy to use. The **f** tag changes the font:

This paragraph is in Helvetica. (We had to spell out "Helvetica" completely—don't forget to do that).

We can also change the type size: from small to **very large** with the **z** tag. Of course, we'd probably also want to change the leading at the same time, but that tag is still to come...

You can reset things back to normal, either by explicitly changing the setting back to its previous value (as we did earlier), or with the **\$** parameter, like so, which restores the font to that specified by the current paragraph's style sheet.

Xtags also lets you use a relative value for many parameters. Here, for example, we increase the current size by 2 points.

Note that these tags can be combined with character format tags, as in `<f"Helvetica"b>H<z8->2<z$->O`. The initial tag sets the font to Helvetica bold, and the subsequent pair of tags creates an eight-point subscript: **H₂O**.

Character Color, Shade, and Opacity Tags

The **c**, **s**, and **p** tags control character color, shade, and opacity, respectively:

<code><c"color name"></code>	Set color, by "color name" or C M Y K W
<code><sshade></code>	Set shade, in percentage of intensity
<code><popacity></code>	Set opacity, as a percentage (<i>QuarkXPress 7.x only</i>)

The **c** tag sets the current text color to the specified document color name. You can abbreviate this to `<cX>` for the following colors: **K** (black), **C** (cyan), **M** (magenta), **Y** (yellow), and **W** (white). For example, applying magenta to the current text can be abbreviated to `<cM>`.

The **s** tag sets the shade of the current text color: its parameter specifies the percentage desired (minimum 0% (no color); maximum 100% (full color); default 100%).

The **p** tag sets the opacity of the current text (QuarkXPress 7.x only): its parameter specifies the percentage desired (minimum 0% (transparent); maximum 100% (opaque); default 100%).

Example 4.3

Xtags Code:

```
@Example: The color <B>c<B> tag takes either a color name
or a built-in color code as its parameter. Turning the
color to white will make the text <cW>invisible<c"Black">.
We'll put a cyan rule behind the next paragraph using a
style sheet so you can see the white type:¶
@Bar:<BcW>Here it is!<PcK>¶
@$:(We'll show you how to make the reverse rule with tags
in Chapter 5.)¶
The shade tag (<B>s<B>) controls the percentage shade of
the text color. For example, <z14B><s50>here is some big
50% Black text enlarged to 14 pt.¶
We can also reduce the visibility by <p50>setting the
opacity to just 50% <Pz$$s$P$>¶
```

Formatted Result:

The color **c** tag takes either a color name or a built-in color code as its parameter. Turning the color to white will make the text `<cW>` invisible. We'll put a cyan rule behind the next paragraph using a style sheet so you can see the white type:

Here it is!

(We'll show you how to make the reverse rules with tags in Chapter 5.)

The shade tag (**s**) controls the percentage shade of the text color. For example, `<z14B><s50>` here is some big 50% Black text enlarged to 14 pt.

We can also reduce the visibility by setting the opacity to just 50%

Character Scaling, Kerning, and Tracking Tags

There are tags to control character scaling, kerning and tracking:

<code><hscale></code>	Set horizontal scaling, in percentage of size
<code><yscale></code>	Set vertical scaling, in percentage of size
<code><kkern></code>	Set kerning for following two characters, in $\frac{1}{200}$ ems
<code><k"Optical"></code>	Set optical auto-kerning method. (<i>InDesign only</i>)
<code><k"Metrics"></code>	Set metrics auto-kerning method. (<i>InDesign only</i>)
<code><ttrack></code>	Set tracking, in $\frac{1}{200}$ ems

InDesign Caveat: Note that in QuarkXPress, **hscale** and **yscale** are mutually exclusive—you can only scale type in one direction. InDesign can accommodate both horizontal and vertical scaling at the same time, but we've limited it to mimic the QuarkXPress implementation. This may change in a future version.

The **h** tag sets the horizontal text scale percentage to *scale* (minimum 25%; maximum 400%; default 100%), and the **y** tag similarly sets the vertical text scale percentage (same limits and default).

The **k** tag sets the kerning between the following character and its right neighbor to *kern* in units of $\frac{1}{200}$ of an em (minimum: -500; maximum: 500). A negative value kerns in (brings letters closer together); a positive value kerns out (pushes letters farther apart).

The **t** tag sets the tracking to *track* in units of $\frac{1}{200}$ of an em (minimum and maximum values are the same as for kerning). A negative value for *track* sets tighter tracking, and a positive value sets looser tracking.

Example 4.4

Xtags Code:

```
@Example: The <B>t<B> tag controls tracking: <t100>here is
some text with loose tracking, <t-15>and here is some with
tighter tracking than normal.<t-0>¶
Here's a kern between the "W" and the "a" (note how the
kerning is applied only to the first character of the pair
to be kerned):¶
@Example Last:<z24><k-20>W<k$>ater.<z11> And not
kerned: <z24>Water.<z11> You can also <h175>scale<h100>
<y220>text<h100>.¶
```

Formatted Result:

The **t** tag controls tracking: h e r e i s s o m e t e x t w i t h l o o s e t r a c k i n g , and here is some with tighter tracking than normal.

Here's a kern between the “W” and the “a” (note how the kerning is applied only to the first character of the pair to be kerned):

Water. And not kerned: Water. You can also *scale text*.

Character Baseline Shift Tags

There are two tags that control character baseline shift:

<bshift> Set absolute baseline shift, in points
<brshift> Set relative baseline shift, in percentage of size

The **b** tag sets the current baseline shift to *shift* points. A positive parameter value moves text above the current baseline, and a negative value moves it below the current baseline. The largest allowed baseline shift in either direction (negative or positive) is three times the current text point size.

The Xtags **br** tag lets you express the baseline shift up or down as a percentage of the current point size, with 0 being no baseline shift, and positive percentages meaning shift upwards, and negative percentages meaning shift downward. (The maximum shift is 300% in either direction.)

Both **<b\$>** and **<br\$>** restore the default baseline shift setting.

Example 4.5

Xtags Code:

@Example: Here are some examples. The words in the **<b-2>**middle of this**<b\$>** line fall 2 points below the normal baseline.¶
 The words in the **<b4>**middle of this**<b\$>** line are 4 points above the baseline.¶
 The next line uses the baseline shift and font size tags to create an exponent in two different ways:¶
<*C>E = mc**<z8>****<b2.5>****<z11>****<b0>** E = mc**<+>****<+>**¶
<*J>In the first instance that we used a relative value for the font size tag's parameter; it said to decrease the current size by 3 points, then shift the baseline up 2.5 points. In the second instance we used a superscript code.¶

@Example Last: Finally, these tags will shift the baseline up by 25% of the font size, which is **<z12>****<br25>**3 points**<z\$br\$>**, in this case.¶

Formatted Result:

Here are some examples. The words in the middle of this line fall 2 points below the normal baseline.

The words in the middle of this line are 4 points above the baseline.

The next line uses the baseline shift and font size tags to create an exponent in two different ways:

$$E = mc^2 \quad E = mc^2$$

In the first instance that we used a relative value for the font size tag's parameter; it said to decrease the current size by 3 points, then shift the baseline up 2.5 points. In the second instance we used a superscript code.

Finally, these tags will shift the baseline up by 25% of the font size, which is 3 points, in this case.

Character Language Tag

The **<ncountrycode>** tag (*QuarkXPress 7.x only*) will set the language attribute of a character run (used for hyphenation and spell-checking), where *countrycode* is one of (note that this is only a partial list):

0	US	1	France	2	Britain
3	Germany	4	Italy	5	Netherlands
6	Flemish	7	Sweden	8	Spain
9	Denmark	10	Portugal	11	French Canada
12	Norway	13	Israel	14	Japan
15	Australia	16	Arabic	17	Finland
18	French Swiss	19	German Swiss	20	Greece

InDesign Caveat: InDesign does not currently support the **n** tag. It will be added in a future release.

Character Ligatures Tag

The **G** tag (*QuarkXPress 7.x only*) will either enable or disable the ligatures attribute of a character run. Valid options are:

<G0>	Disable ligatures
<G1>	Enable ligatures
<G\$>	Set ligatures to paragraph style
<G\$\$>	Set ligatures to character style

Special Character Tags

The **** tag is used to place special characters into formatted text:

<\special>

Special can take on many forms. Tags marked with a † may be made non-breaking if the backslash is followed by an exclamation point (!). For example, the tag **<!s>** places a non-breaking space into the document:

<\n>	Insert line break (“soft return”, not paragraph return)
<\d>	Insert discretionary (optional) line break
<\->	Insert hyphen
<\m>†	Insert em space (<v7.00> tag mode only);
	Insert m dash (tags prior to <v7.00>)
<\i>	Insert “indent here” marker
<\t>	Insert right indent tab (not a regular tab—see below)
<\s>†	Insert standard space
<\f>†	Insert flex space (<v7.00> tag mode only);
	Insert figure (en, half-em) space (tags prior to <v7.00>)
<\p>†	Insert punctuation space
<\q>†	Insert quarter-em (flexible) space
<\h>†	Insert discretionary (optional) hyphen
<\2>	Insert previous text box page number
<\3>	Insert current text box page number
<\4>	Insert next text box page number
<\c>	Insert new column (force column break)
<\b>	Insert new box (force box break)
<\@>	Insert at sign (@)
<\<>	Insert left angle bracket (<)
<\>>	Insert right angle bracket (>)
<\\>	Insert backslash (\)
<\e>	Insert “End Nested Style” character. (<i>InDesign only</i>)

<\#nnn>† Insert special character, where *nnn* is a decimal value representing a character from either MacRoman (if the encoding is set to **<e0>**) or WinLatin (if the encoding is set to **<e1>**). If the encoding is neither **<e0>** nor **<e1>** (see *<e>* tag on page 22), then the character set will be selected from the platform default. The most common uses for this tag are **<\#13>** (paragraph return) and **<\#9>** (tab)

<\#Unnnn>

<\#U+nnnn> Insert Unicode special “character,” where *nnnn* is a 4-digit hexadecimal character code point like **<\#U2122>** (the ™ symbol) or **<\#U+20AC>** (the Euro symbol)

Note that the **<\t>** tag does not insert a regular tab character; instead, use **<\#9>** or a normal tab character (character code 9).

The following tags were introduced in QuarkXPress 7.x, and will fail with “No such tag” on any other platform (except for the **<\e>** tag, which has an alternate meaning in InDesign, see above):

<\e>†	Insert en space (half-em)
<_>†	Insert breaking em dash
<\5>†	Insert 3-per-em space
<\\$>†	Insert 4-per-em space
<\^>†	Insert 6-per-em space
<\8>†	Insert figure space
<\[>†	Insert thin space
<\{>†	Insert hair space
<\j>†	Insert word joiner

InDesign Caveat: Since InDesign won’t break on special space characters (en space, flex space, punctuation space, etc.), like QuarkXPress does, Xtags follows any such character with a zero-width space (on which InDesign will break) so long as non-breaking (!) hasn’t been specified. For example, a **<\q>** tag now generates two characters, a four-per-em-space followed by a zero-width-space, while a **<!q>** tag will still generate just a four-per-em-space character with the no-break attribute applied.

Character Set Encoding Tag and XPress Tags Version Tag

The **e** tag selects the character set to use for subsequent input:

<en>

n is a number indicating the desired character set:

<e0>	Mac OS character set
<e1>	Windows dtp character set
<e2>	ISO Latin-1
<e8>	UTF-16
<e9>	UTF-8

The default is **<e0>** under Mac OS and **<e1>** under Windows. UTF-16-encoded files are always auto-sensed and any **<en>** tag in them is ignored. UTF-8 looks like a normal 8-bit encoding, so Xtags needs a hint: either the file should start with a byte order mark (BOM) or it must contain the tag **<e9>** very early in the file (ideally, in the first paragraph) in order to be recognized.

Any normal content of the paragraph containing the **e** tag will be ignored, and thus the following paragraph-ending character will have no effect on the output.

The **v** tag specifies the desired version of XPress Tags under which to interpret subsequent tags:

<vn.m>

n and **m** denote the major and minor version numbers. The default as of this writing is **<v2.00>**, except for QuarkXPress 7.x, which defaults to **<v7.00>**.

Any normal content of the paragraph containing the **v** tag will be ignored, and thus the following paragraph-ending character will have no effect on the output.

Output from Xtags and XPress Tags usually begins with the line:

<en><vn.m>¶

which doesn't introduce a blank line in the input because both tags cause the containing paragraph to be completely ignored.

Xtags for InDesign and Xtags under QuarkXPress 6.x always output **<v2.00>** tags. Under QuarkXPress 7.x, Xtags outputs **<v7.00>** tags.

Xtags will output tags in the native character set (MacRoman under Mac OS X, WinLatin under Windows) when running under QuarkXPress 6.x. Xtags for InDesign and Xtags under QuarkXPress 7.x allow you to select the target character set, defaulting to UTF-8.

Chapter 5

Setting Paragraph Attributes

There are several tags designed to control paragraph formatting. We begin with the simplest of them: those that control paragraph alignment.

Paragraph Alignment Tags

<code><*L></code>	Left-aligned paragraph
<code><*C></code>	Center-aligned paragraph
<code><*R></code>	Right-aligned paragraph
<code><*J></code>	Justified paragraph
<code><*F></code>	Force-justified paragraph

Example 5.1

Xtags Code:

```
@Example:<*L>This paragraph is left-aligned.¶
<*C>This paragraph is centered.¶
<*R>This paragraph is right-aligned.¶

<*J>This paragraph is justified, which means we have to
have enough text here to show the justification.¶

@Example Last:<*F>This paragraph is force-justified, which
means we have to have enough text here to show the forced
justification, which otherwise wouldn't be visible at all
with just a short line.¶
```

Formatted Result:

This paragraph is left-aligned.

This paragraph is centered.

This paragraph is right-aligned.

This paragraph is justified, which means we have to have enough text here to show the justification.

This paragraph is force-justified, which means we have to have enough text here to show the forced justification, which otherwise wouldn't be visible at all with just a short line.

Paragraph Basic Settings Tag

The `*p` tag controls a variety of paragraph settings. It has this general format:

```
<*p(left indent, first indent, right indent, leading,
space before, space after, lock to grid, language)>
```

The parameters to `*p` are:

left indent

Left margin (minimum 0 pt; maximum is the current text column width; default is the current paragraph's left indent setting).

first indent

First line indent (minimum is the negative of the left margin setting; maximum is the current text column width; default is the current paragraph's first indent setting).

right indent

Right margin (minimum 0 pt; maximum is the current text column width; default is the current paragraph's right indent setting).

leading

Leading (minimum 0 pt; maximum is the current text box height; default is the current paragraph's leading setting). Placing a plus sign before this parameter specifies incremental leading; making this parameter zero requests "auto leading" (see the QuarkXPress manual for a full discussion of the various types of leading).

InDesign Caveat: Incremental (relative) leading is not supported natively by InDesign, so Xtags converts the incremental value to an absolute value during import. Therefore, the leading of text to which incremental leading has been applied is not "live" and will not change if the type size is altered later.

InDesign Caveat: Leading and language are paragraph attributes in pre-version QuarkXPress 7.0 tags but are character attributes in InDesign. When exporting Xtags text, the corresponding parameters of the `<*p ()>` tag are filled only if the entire paragraph uses the same leading or language value.

If the leading or language setting changes within the paragraph, the corresponding `<*p()` parameter will be left empty.

space before

Vertical space before the paragraph (minimum 0 pt; maximum is the current text box height; default is the current paragraph's space-before setting).

space after

Vertical space following the paragraph (minimum 0 pt; maximum is the current text box height; default is the current paragraph's space-after setting).

lock to grid

A one-character code indicating whether or not to lock the lines of the current paragraph to the baseline grid (use **G** to lock to the grid, **g** to unlock from the grid; default is the current paragraph's lock-to-grid setting).

language

The language to be used for hyphenation and spell-checking. This parameter only exists for backward-compatibility and is ignored for `<v7.00>` tags. Please use the `<n>` tag (see page 21) to set the language attribute in QuarkXPress 7.x.

Example 5.2

Xtags Code:

```
@$:Here are some examples: <*p(.25", , .25")>This paragraph
has quarter-inch margins on both sides.¶
<*p(, 12)>Omitted settings don't change, so this paragraph
has 1/4" margins as above, plus a 12 point indent on the
first line.¶
@$:<*p(0, 0, 0, 16)>This paragraph has the same settings
as the <f"Helvetica"z9B>Normal<f$z$> style, but with the
leading set to 15 points.¶
@$:<*p(, , .+12,,,.-12)>Relative parameter values are
supported too. For example, this paragraph has extra space
at the right margin and no space after it.¶
@$:<*p(.+24,,,.+24,,$)>Similarly, this paragraph's margins
are both widened by 24 points (and the default space after
setting is restored).¶
<*p(2.5p, -1.5p, 0, 1p)><f"Zapf Dingbats">v<f$>→This
paragraph creates a hanging indent by setting the first
line indent to a negative value.¶
```

Formatted Result:

Here are some examples: This paragraph has quarter-inch margins on both sides.

Omitted settings don't change, so this paragraph has 1/4" margins as above, plus a 12 point indent on the first line.

This paragraph has the same settings as the Normal style, but with the leading set to 15 points.

Relative parameter values are supported too. For example, this paragraph has extra space on the right margin and no space after it.

Similarly, this paragraph's margins are both widened by 24 points (and the default space after setting is restored).

- ❖ This paragraph creates a hanging indent by setting the first line indent to a negative value.

Paragraph Tab Settings Tag

The `*t` tag sets any number of tab stops in the current paragraph using the following format:

```
<*t(position1, alignment1, "fill1", position2, alignment2,
"fill2", ...) >
<*t0> or <*t()> turns off all tabs
```

One tab stop is set at each specified *position_n*, with alignment *alignment_n*, having a fill character of *fill_n*.

Tab alignments are specified using one of the the following codes:

0	left align tab
1	centered tab
2	right align tab
4	decimal-aligned tab
5	comma-aligned tab
"x"	align on the specified character (where x is any character, e.g., "/" means align on slashes). Note that the surrounding quotation marks must be included.

Note that QuarkXPress takes internationalization into account, so that comma and decimal-aligned tabs (codes **5** and **4**) will be localized to the appropriate equivalent. Thus, to create a tab aligned on a comma character, when comma-aligned tabs are defined in some other way, use `" , "`. Decimal alignment should similarly be defined as `" . "` when you want that specific non-localized behavior.

To set no tabs, use the form `<*t()>` or `<*t0>`.

Fill characters have the form "1xx" or "2xy" where xx and xy indicate the desired fill character(s). You must include the quotation marks. The 1 form specifies a single fill character (repeated twice), and the 2 form specifies two alternating fill characters. For example, "1.." specifies a period as the fill character, and "2. " specifies a fill composed of alternating periods and spaces (beginning with the former).

Use "1 "—the number one followed by two space characters—to signify no fill (which is also the default if this parameter is omitted).

Example 5.3

Xtags Code:

```
@Example:<*t(24,0,"1 ",144,1,"1 ",246,2,"2 ." ,326,4,"2 ")>¶
→45-102-P→Aquastar Pro→40%→$8.75¶
→48-103-H→Aquaseal Deluxe→44%→$128.99¶
→45-202-Q→Aquamaster Underwater→48%→$.75¶
```

Formatted Result:

<i>Aligned left at 24 pts</i>	<i>Aligned Center at 144 pts</i>	<i>Aligned Right at 246 pts, with dot leader</i>	<i>Decimal Aligned at 326 pts, with dot leader</i>
45-102-P	Aquastar Pro40%.....	\$8.75
48-103-H	Aquaseal Deluxe44%.....	\$128.99
45-202-Q	Aquamaster Underwater48%.....	\$.75

Paragraph Hyphenation and Justification Tag

The `*h` tag sets the hyphenation and justification scheme for the current paragraph:

```
<*h" h&j name">
```

where `h&j name` is the name of a defined H&J scheme.

InDesign Caveat: H&Js are handled very differently in InDesign. There are no global H&J styles, and all H&J settings in InDesign are assigned at the paragraph level, so this tag is ignored.

Paragraph Rules Tags

The `*ra` and `*rb` tags specify a horizontal rule above and below the current paragraph, respectively.

```
<*ra(thickness, style, color, shade, [opacity,] from left,  
from right, offset)>  
<*rb(thickness, style, color, shade, [opacity,] from left,  
from right, offset)>  
<*ra0> Turn off rules above  
<*rb0> Turn off rules below
```

The `*ra` and `*rb` tags have identical parameters:

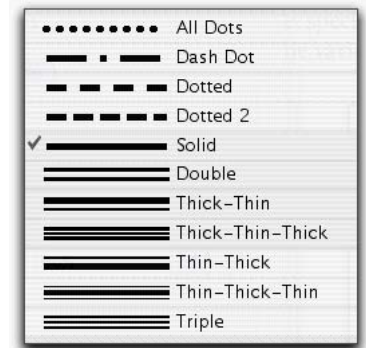
thickness

Vertical thickness of the rule (minimum 0 pt; maximum is 15"; default 1 pt).

style

A numeric code specifying the line style:

- 0 = solid (*the default style*)
- 1 = short dashes
- 2 = long dashes
- 3 = dot-dashes
- 4 = all dots
- 128 = double line (medium, medium)
- 129 = double line (thin, thick)
- 130 = double line (thick, thin)
- 131 = triple line (thin, thick, thin)
- 132 = triple line (thick, thin, thick)
- 133 = triple line (thin, thin, thin)



color

Rule color (default Black).

shade

Rule shade (minimum 0%; maximum 100%; default 100%).

opacity (`<v7.00>` tags only)

The *opacity* parameter (minimum 0%; maximum 100%; default 100%) only exists for tags marked as `<v7.00>` (see page 23). It must be omitted otherwise; consequently, the `<v7.00>` `*ra` and `*rb` tags are not backward-compatible with previous versions.

from left

Indent from the left margin (minimum 0 pt; maximum is the width of the current text column; default 0 pt). If this value is prefixed with a `T`, then both *from left* and *from right* indents are "text-relative," i.e., relative to the left and right horizontal extents of the top (`*ra`) or bottom (`*rb`) text line.

from right

Indent from the right margin (minimum 0 pt; maximum is the width of the current text column; default 0 pt; see note immediately above about text-relative indents).

offset

Offset from the text baseline (up is implied by a positive value), either in absolute terms (a distance; minimum is the negative of one-half the rule thickness; maximum is the height of the current text box; no default) or in relative terms (a percentage of the space between the previous (**ra*) or next (**rb*) paragraph and the current paragraph; minimum 0%; maximum 100%; default 0%).

InDesign Caveat: QuarkXPress lets you do relative rule positioning, where you assign a percentage (rather than an absolute value) to the rule offset. It takes the space between paragraphs and places the rule between the two at the percentage you specify. If you're using percentage offsets and are using thick rules, it will also create a buffer zone so that the rule doesn't overprint on the paragraph on the other side of the rule. Neither of these options is available in InDesign. You can, however, use space above and space below to create a similar effect.

The special tag forms `<*ra0>` and `<*rb0>` turn off the corresponding rule altogether.

The following are examples of the paragraph rule tags:

Example 5.4

Xtags Code:

```
@Example:The <B>*ra<B> and <B>*rb<B> tags create rules
above and below the current paragraph, respectively.¶
This first paragraph has no rules associated with it.¶
<*ra(1,"Solid",K,100,100†,0,0,10)>This paragraph has a
one-point rule above it, running the width of the column,
offset 10 points up.¶
<*ra(3,"Solid",K,60.001,100†,36,36,10)><*rb(3,"Solid",K,6
0.001,36,36,0%)><I>Notice:<I> This paragraph has thicker
60% grey rules above and below its central area. It's also
indented .5" from the left and right.¶
<*ra0><*rb(2,"All Dots",K,100,100†,T0,0,0%)>This
paragraph's rule below is only as wide as its last line,
so we have to have a long enough paragraph to show this
effect. Also notice that it's using a rule style that's
not solid.¶
```

```
<*rb0><*ra(16,"Solid",K,100,100†,0,0,-4.5)><B><cW>This bold
text is reversed (white), in front of a black rule.<B><cK>¶
```

Note: Opacity parameters are marked with †, and only exist for `<v7.00>` tags.

Formatted Result:

The **ra* and **rb* tags create rules above and below the current paragraph, respectively.

This first paragraph has no rules associated with it.

This paragraph has a one-point rule above it, running the width of the column, offset 10 points up.

Notice: This paragraph has thicker 60% grey rules above and below its central area. It's also indented .5" from the left and right.

This paragraph's rule below is only as wide as its last line, so we have to have a long enough paragraph to show this effect. Also notice that it's using a rule style that's not solid.

This bold text is reversed (white), in front of a black rule.

Paragraph Drop Caps Tag

The **d* tag allow you to create a drop cap in the current paragraph:

```
<*d(chars, lines)>
```

The *chars* parameter defines how many characters will be modified to be drop caps (minimum 1, which is the default). The *lines* parameter defines how many lines deep the drop cap extends to (minimum 2; maximum in QuarkXpress is 16, maximum in InDesign is 25). The default is 1, which effectively means no drop caps.

Example 5.5

Xtags Code:

```
@$:The <B>*d<B> tag creates drop caps. Here are three
examples:¶
<*d(1,2)><k20>N<k0>otice that this paragraph has some kerning
between the drop cap the the second letter. This isn't always
needed, but it's obvious when it is<\m>the letters crash and
the type looks very crunched together.¶
```

```
<*d(1,2)><z15>H<z11>ere's another way to vary the drop
cap<\m>by increasing the size of the letter. It's now
functioning as a combined drop cap and stick-up initial. No
kerning needed between the drop cap and text this time.¶
<*d(5,2)>Drop caps can also consist of more than one
character, as this example shows. If you're going to use
the whole word, count the letters of the word and include
the space after the word. That extra space gives you a nice
transition to the text. Descenders often cause problems in
cases like this.¶
```

Formatted Result:

The `*d` tag creates drop caps. Here are three examples:

Notice that this paragraph has some kerning between the drop cap the the second letter. This isn't always needed, but it's obvious when it is—the letters crash and the type looks very crunched together.

Here's another way to vary the drop cap—by increasing the size of the letter. It's now functioning as a combined drop cap and stick-up initial. No kerning needed between the drop cap and text this time.

Drop caps can also consist of more than one character, as this example shows. If you're going to use the whole word, count the letters of the word and include the space after the word. That extra space gives you a nice transition to the text. Descenders often cause problems in cases like this.

form, `<*kt(A)>`, makes all lines in a paragraph be kept together (i.e., appear in a single column).

The second form specifies how the text layout process may break the current paragraph over text columns. *Start lines* specifies the minimum number of lines that should be kept together (without breaking) at the start of the paragraph (minimum 1; maximum 50; default 2), and *end lines* specifies how many lines should be kept together at the end of the paragraph (minimum 1; maximum 50; default 2).

The final form, `<*kt0>`, turns off all widow and orphan control.

Paragraph “Keep Together” and “Keep With Next” Tags

The `*kt` and `*kn` tags allow you to set paragraph “keep together” and “keep with next” attributes:

```
<*knonoff>
<*kt(A)>
<*kt(start lines, end lines)>
<*kt0>
```

The `*kn` (“keep with next”) tag specifies whether to keep at least part of the current paragraph with the next, across column breaks. Its parameter *onoff* must be either a **1** (keep with next) or a **0** (don't keep with next).

The `*kt` (“keep together”) tag controls “widows” and “orphans” (roughly, lines in a paragraph left by themselves at the bottom or top of a text column—eminent modern typographers disagree on the exact definition of these terms). The first

Example 5.6

Xtags Code:

```

@Example:<*L><*p(0,0,0,13,0,0,g,)>We've changed our normal
example layout to illustrate these tags more clearly. The
<B>*k<B> tags specify paragraph keep options. <B>*kn1<B>
says to keep the current paragraph (or at least part of
it) with the next one.<*kn1><*kt0>¶
<*p(,12)>Inserting a <@Xtags Tags><\>*kn1<\><@$> <@$p>tag
in the previous paragraph will force the last line of the
previous paragraph to keep with this paragraph (it ends up
being 2 lines in this text box, however). If we had also
used a <@Xtags Tags><\>*kt(1,2)<\><@$p> tag, it would
have only left one line of the first paragraph. If we had
used a <@Xtags Tags><\>*kt(2,2)<\><@$p> tag (keep the
first 2 and last 2 lines together), that first box would
be empty, and text would start in the second box. Also
notice the <@Xtags Tags><\>*kt0<\><@$p> tag; this turns
off "keep lines together", a normal part of the <@Menu/
Dialog Item>Example<@$p> paragraph style.¶
The <B>*kt<B> tag controls how many lines of a paragraph
must be kept together when it begins and ends in different
columns, boxes, or pages. <*kt(5,4)>This paragraph must
keep 5 lines starting together and 4 lines ending together.
The box above has enough space for 4 lines of text,
but it can't start there because of the 5 line minimum
requirement. Similarly, there's room a the bottom of the
middle column for an additional line 3 lines of text, but
because 4 lines are requires to end the paragraph, it
can't use them.¶
Finally, this last paragraph's lines must all remain
together <*kt(A)>within one column, which is why it jumps
to the last box. If this last box weren't here, the text
would over-set.¶

```

Formatted Result:

We've changed our normal example layout to illustrate these tags more clearly. The *k tags specify paragraph keep options. *kn1 says to keep the

current paragraph (or at least part of it) with the next one.

Inserting a <*kn1> tag in the previous paragraph will force the last line of the previous paragraph to keep with this paragraph (it ends up being 2 lines in this text box, however). If we had also used a <*kt(1,2)> tag, it would have only left one line of the first paragraph. If we had used a <*kt(2,2)> tag

(keep the first 2 and last 2 lines together), that first box would be empty, and text would start in the second box. Also notice the <*kt0> tag; this turns off "keep lines together", a normal part of the **Example** paragraph style.

The *kt tag controls how many lines of a paragraph must be kept together when it begins and ends in different columns, boxes, or pages. This paragraph must keep 5 lines starting together and 4 lines ending together. The box above has enough

space for 4 lines of text, but it can't start there because of the 5 line minimum requirement. Similarly, there's room a the bottom of the middle column for an additional line 3 lines

of text, but because 4 lines are requires to end the paragraph, it can't use them.

Finally, this last paragraph's lines must all remain together within one column, which is why it jumps to the last box. If this last box weren't here, the text would over-set.

Chapter 6

Defining and Applying Style Sheets

We've already seen style sheet application tags in action. In this chapter we will examine them more formally.

Defining Paragraph Style Sheet Tags

Either of the following tag forms can be used to define paragraph style sheets:

```
@name=[S]<tags>¶
@name=[S"based-on", "next", "char-style-sheet"]<tags>¶
```

You define a paragraph style sheet with an “at sign” (@), followed by the name of the style sheet, an equals sign, a prefix ([S] or [s]) indicating a style sheet definition, and then the tags for the desired attributes.

Spaces are very significant in the style sheet name. For example,

```
@ spaces galore =<...>
```

does *not* define the same style as

```
@spacesgalore=<...>
```

Leading and trailing spaces and multiple embedded spaces in style names are best avoided.

Only the first definition of a style sheet has any effect. Xtags ignores any attempts to re-define a style sheet having the same name as one you've already defined. Similarly, if you try to define a style sheet that is already a part of the document into which you're importing text, Xtags ignores the attempt. This has an important ramification for “debugging” your Xtags sources. When you import a file and discover an error in a style sheet definition, the natural thing to do is to delete the erroneous text, modify the text file, and then import again. However, you must also delete the style sheet definitions for any style sheets defined by the imported file before reimporting it, or revert your document to the last saved state. Other-

wise, the new definitions in the file will be ignored, the old definition will remain in effect, and the error will seem to persist despite your changes.

You may optionally specify a “based-on” style as part of the style definition prefix, as in [S“based-on”], where *based-on* is the name of the style sheet upon which to base this new style sheet. The based-on style must already be defined. If no based-on style is given, then the style sheet being defined is based on [Basic Paragraph] in InDesign, or No Style in QuarkXPress, though all its settings are initialized from the **Normal** style (which is guaranteed to be present in every XPress document).

You may also optionally specify a next-style sheet setting as the second parameter to **s**. The default is the style sheet itself.

Finally, you may optionally specify a character style sheet as the tag's third parameter. The specified character style sheet must already exist. The character style sheet affects character level formatting only, not paragraph level formatting.

InDesign Caveat: InDesign paragraph styles don't support an associated character style. So tags like <@\$p:> (get the character style from the current paragraph styles) will call in [None].

Any relative parameter values in <tags> are computed relative to the based-on style (or to the settings copied from **Normal** or [Basic Paragraph] if there is no explicit based-on style). It is impossible to define or redefine the **Normal** or [Basic Paragraph] style with Xtags, since it can not be deleted from any document.

Defining Character Style Sheet Tags

Character style sheets are defined similarly to paragraph style tags:

```
@name=[S"", "", "", "based-on-char-style"]<tags>
@name=<tags>
```

where *based-on-char-style* is the existing character style sheet upon which the new one is to be based. The default is no based-on character style sheet. (The second tag form is equivalent to the first form with the *based-on-char-style* parameter omitted.)

Applying Style Sheet Tags

A paragraph style sheet is applied by starting the paragraph with an “at” symbol, followed by the name of the style sheet, and concluded with a colon, like this:

@paragraphstylesheetname: A paragraph style sheet should generally start at the beginning of the paragraph, but it can appear anywhere before the end of the paragraph (the final break character).

Character style sheets are encased in angle brackets and start with the “at” sign, like this: `<@characterstylesheetname>` tag. A character level style sheet can start anywhere in the paragraph.

When a style sheet is applied (either paragraph or character), all the properties of the named style sheet take effect at the current point of input; the style sheet association is also remembered at the paragraph level. The applied style sheet will remain in effect, for succeeding paragraphs, until another style sheet is applied. Under Xtags, if more than one style sheet is applied within a paragraph, the final one takes precedence.

DESIRED RESULT	PARAGRAPH STYLE SHEET	CHARACTER STYLE SHEET
<i>Apply specified style sheet</i>	<code>@name:</code>	<code><@name></code>
<i>Apply the default (Normal or [Basic Paragraph]) style sheet</i>	<code>@\$:</code>	<code><@\$></code>
<i>Apply character style of current paragraph style</i>	<code><@\$p></code>	
<i>Apply “No Style”</i>	<code>@:</code>	<code><@></code>

The forms `@$:` and `<@$>` are shorthand ways of applying the **Normal** (InDesign applies **[Basic Paragraph]** if **Normal** does not exist) paragraph or character style sheet, respectively (the “longhand” ways being `@Normal:` and `<@Normal>`). The forms `@:` and `<@>` are the only method for applying the **No Style** (**[None]** in InDesign) paragraph and character style sheet, respectively.

If a style sheet is applied that has not been defined, then it will be created as a copy of the **Normal** style, based on **No Style** (**[Basic Paragraph]** in InDesign), and applied normally.

Note that no extraneous spaces should be used in or around the style sheet name, nor should any appear before the `@` (nor after the final colon in the case of a paragraph style sheet). For example, normally

```
@astyle:This is the first line of a paragraph with style
“astyle.”¶
```

would be correct, while

```
@a style: We’re making two mistakes in this example.¶
```

has two likely problems. First, there is a space in the style name itself (which will cause a new style with that name to be created), and second, there’s space after

the colon (which means that there will be some undesired space at the beginning of the paragraph).

Example 6.1

Xtags Code:

```
@Menu/Dialog Item=<B><s100><t0><h100><z11><k0><b0><cK>
<f"MinionPro-Regular">
@Indent Character Style=<B><s60.001><t0><h100><z9><k0><b0>
<c"Blue"><f"Helvetica">
@Rules Character Style=[S"", "", "", "Indent Character
Style"]<z8><c"Red">
@B-22=<P><s100><t0><h100><z11><k0><b0><cK><f"MinionPro-
Regular">
@Indent=[S"Example", "Indent", "B-22"]<*p(36,0,36,13,0,12,g,
"U.S. English")>
@Rules Above And Below=[S"Indent", "Example Last"]<*ra(3,
"Solid", K, 50, 100†, 0, 0, 8)><*rb(3, "Solid", K, 50, 100†, 0, 0, 4)><*
p(36,0,36,10,0,12,g, "U.S. English")><I><z9><f"Helvetica">
@Example=[S"", "Example", "B-22"]<*J><*h"Standard"><*kn0>
<*kt(2,2)><*ra0><*rb0><*d0><*p(12,0,12,13,0,6.5,g, "U.S.
English")>
@Example Last=[S"Example", "Example Last", "B-22"]<*rb(1.5,
"Solid", "PANTONE 569 C", 50, 100†, -12, -
12,9)><*p(12,0,12,13,0,26,
g, "U.S. English")>
@Example:Here is the primary style sheet we’re using
in these examples, which we’ve called <@Menu/Dialog
Item>Example<@$p>. It uses very common settings, including
the <@Menu/Dialog Item>B-22<@$p> character style sheet.
@Indent:This paragraph uses another style sheet, <@Indent
Character Style>Indent<@$p>, based on <@Indent Character
Style>Example<@$p>, that brings in both margins a bit. Note
that we use the character style sheet <@Indent Character
Style>Indent Character Style<@$p> to format style sheet
names in this example.
@Rules Above And Below:This style sheet named <@Rules
Character Style>Rules Above And Below<@$p> is based on
<@Rules Character Style>Indent<@$p>. It retains that
style sheet’s margins and adds rules above and below the
paragraph. It also changes the font family, size, and
style to nine point italic Helvetica type and specifies <@
Rules Character Style>Example Last<@$p> as its next style
setting. We use the character style sheet <@Rules Character
Style>Rules Character Style<@$p> to format style sheet
names in this paragraph.Note that we must turn off this
```

character style sheet by reverting to the `paragraph` style sheet's character style.

@Example Last: We define each style sheet at the beginning of the document, each in a separate paragraph. We've used the colon continuation character to extend some style sheet definitions to two lines for readability. Finally, we apply style sheets by placing a paragraph style tag at the beginning of the paragraph.

Note: *Opacity parameters (marked with †) of the `*ra` and `*rb` tags only exist when the tags are marked as `<v7.00>` or greater (see page 23).*

Formatted Result:

Here is the primary style sheet we're using in these examples, which we've called **Example**. It uses very common settings, including the **B-22** character style sheet.

This paragraph uses another style sheet, **Indent**, based on **Example**, that brings in both margins a bit. Note that we use the character style sheet **Indent Character Style** to format style sheet names in this example.

*This style sheet named **Rules Above And Below** is based on **Indent**. It retains that style sheet's margins and adds rules above and below the paragraph. It also changes the font family, size, and style to nine point italic Helvetica type and specifies **Example Last** as its next style setting. We use the character style sheet **Rules Character Style** to format style sheet names in this paragraph. Note that we must turn off this character style sheet by reverting to the **paragraph** style sheet's character style.*

We define each style sheet at the beginning of the document, each in a separate paragraph. We've used the colon continuation character to extend some style sheet definitions to two lines for readability. Finally, we apply style sheets by placing a paragraph style tag at the beginning of the paragraph.

Chapter 7

Creating Text, Picture, None, and Line Boxes

Box Creation using Basic Parameters

Text boxes, picture boxes, none boxes, and lines can be either unanchored or anchored. Unanchored items are independent entities which remain in a fixed position on the page unless you explicitly move them. Anchored items are created by copying or cutting the item (or group of items) and then pasting it into a text flow. The item is treated as another character within the text story, and it flows along with the accompanying text as changes are made to the preceding text flow. Xtags can create either type of item.

InDesign Caveat: Since InDesign doesn't support the automatic alignment of anchored frames to the text ascent, we've added such support to our product. However, this feature is not dynamic; anchored frames will be top-aligned by Xtags only during import. The anchored frame will not automatically re-align to a new ascent if the surrounding text is altered after the import.

InDesign Caveat: Anchored boxes in InDesign have several advanced placement features that are not (yet) supported by Xtags.

Tags for Text Boxes

The **&tb** and **&tbu2** tags create and fill an anchored or unanchored text box, and start a new text flow for the box contents, while the **&te** tag marks the end of either kind of box contents. The **&tb** and **&tbu2** tag parameters are very similar—when a parameter is unique to an anchored or unanchored tag, we've underlined it. Parameters whose names are shown in **bold** also have more advanced, expanded forms which are covered in greater detail in later subsections. Parameters generally take their default values from the document's rectangular text box

tool, but those marked with a † have no tool defaults (see the parameter description for defaults assigned by Xtags).

The **anchored text box tag** with its basic parameters has one of the following forms:

```
<&tb(width†, height†, anchored alignment†, frame width,
frame color, frame shade, frame style, background color,
background shade, text outset, columns, gutter, text inset,
baseline offset, baseline minimum, vertical alignment,
interparagraph maximum†, box name†)>...Text to be placed in the box,
typically containing other tags...<&te>
```

The **&tb2** tag adds additional parameters (**box angle**, **box skew**, **flags**, and **item runaround**), and is otherwise identical:

```
<&tb2(width†, height†, box angle†, box skew†, flags†,
anchored alignment†, item runaround, frame width, frame
color, frame shade, frame style, background color,
background shade, text outset, columns, gutter, text inset,
baseline offset, baseline minimum, vertical alignment,
interparagraph maximum†, box name†)>...Text to be placed in the box,
typically containing other tags...<&te>
```

There are two things to keep in mind regarding anchored text boxes.

- When an anchored text box is created, all text settings that were in effect beforehand remain in effect for text flowing into the box. In other words, text within an anchored text box inherits all character and paragraph attributes in effect at the time of the box's creation. However, any changes made within an anchored text box do not persist when the box is finished (i.e., when **&te** is encountered). All character and paragraph settings in effect when a box is created are saved by Xtags and restored at the point of the **&te** tag.
- You can't nest anchored boxes in QuarkXPress. That is, an **&tb** tag may not appear inside another **&tb**/**&te** pair. Similarly, the **&pb** (create anchored picture box) tag may not appear inside a **&tb**/**&te** pair. Xtags for InDesign has no such limitation.

The **unanchored text box tag** with its basic parameters has this form:

```
<&tbu2(x†, y†, width†, height†, box angle†, box skew†, flags†,
item runaround, frame width, frame color, frame shade,
frame style, background color, background shade, text
outset, columns, gutter, text inset, baseline offset,
baseline minimum, vertical alignment, interparagraph
```

maximum†, box name†, layer name†)>...Text to be placed in the box, typically containing other tags...<&te>

If XMLxt, Autopage, or XPressMath XTensions are present, each will be invoked on created text boxes after all contents have been processed.

Both the anchored and unanchored text box tags have parameters that can be further expanded as more complex sub-parameters. These include:

x: (*x offset, relative placement, relative box reference, relative box domain*)

width: (*width, sizing specs, adjustment/margin, minimum*)

height: (*height, sizing specs, adjustment/margin, minimum, leading adjustment*)

frame width: (*frame width, corner diameter, corner type*)

frame color: (*frame color, frame gap color*)

frame shade: (*frame shade, frame gap shade, frame opacity, frame gap opacity*)

background color: (*background color, blend color, blend style, blend angle, start position, center position, end position*)

background shade: (*background shade, background opacity, blend shade, blend opacity*)

text outset, text inset: (*top, left, bottom, right*)

Parameters for anchored and unanchored text boxes

x
(*see page 44*) *Unanchored boxes only.* Horizontal page-relative position.

y
Unanchored boxes only. Vertical page-relative position.

width
(*see page 45*) Width of the box (minimum is 2 pt; maximum 3456 pt; default 72 pt).

height
(*see page 46*) Height of the box (minimum is 2 pt; maximum 3456 pt; default 72 pt).

box angle

Tb2 and tbu2 tags only. Specifies the rotation in degrees about the bounding box midpoint, positive being counterclockwise and negative being clockwise (minimum: -360; maximum 360; default 0).

box skew

Tb2 and tbu2 tags only. The angle with which the box is skewed from vertical in degrees, with a negative value skewing the box to the left of vertical (minimum: -75; maximum: 75; default: 0).

flags

Tb2 and tbu2 tags only. An optional string consisting of one or more code or flag letters having the following meanings:

- ◆ **b** (or **B**) = **box** print suppression
- ◆ **p** (or **P**) = **picture** box content print suppression
- ◆ **k** (or **K**) = send the newly-created box behind all other boxes on the spread (as in “send to back”) (*Unanchored boxes only*)
- ◆ **l** (or **L**) = **lock** item
- ◆ **h** (or **H**) = flip box contents **h**orizontally
- ◆ **v** (or **V**) = flip box contents **v**ertically
- ◆ **u** (or **U**) = send box behind (under) its referenced box (The “k” flag will be ignored when both flags are given) (*Unanchored boxes only*)

anchored alignment

Anchored boxes only. Box alignment with respect to the text baseline for anchored boxes. You can optionally use **A** (or **a**) for ascent alignment (where the box grows down from the top of the ascender of the tallest character in its text line, if any) or use **B** (or **b**) for baseline alignment of the box (the default, where the created text box aligns with the baseline of its line as if it were a text character). For baseline offset, you must specify a value between -11” to 11.5”, in the form (*text align, offset*).

item runaround

Tb2 and tbu2 tags only. Either **I** for item runaround or **N** for no runaround (these are case-insensitive, as are most of the single-character keys).

frame width

(*see page 48*) Width of the box’s frame (minimum 0 pt (none); maximum 504 pt).

frame color

(*see page 49*) Color of the box’s frame: either a quoted color name or one of the key characters as described under the <c> tag in Chapter 4.

frame shade

(see page 49) Shade of the box's frame, as a percentage (minimum 0%; maximum 100%).

frame style

Style of the box's frame, specified as the frame style name in quotes (e.g., "Solid"). Following is a list of the various applications and frame style names.



QuarkXPress frame names

InDesign frame names

InDesign Caveat: Xtags for InDesign accepts QuarkXPress algorithmic names as well as any InDesign frame names. QuarkXPress has several bitmapped styles (**Yearbook** through **Op Art2**) that are mapped to InDesign's **Solid** style, as is the **Dash Dot** style (since there's no InDesign equivalent style). English, French, and German-localized names are supported.

background color

(see page 50) Background color of the box, or **N** for no ("None") color.

background shade

(see page 50) Background shade of the box, as a percentage (minimum 0%; maximum 100%).

text outset

(see page 50) Text runaround exterior margin on all four sides of the box (minimum -288 pt; maximum 288 pt). *Use negative values with caution: it appears to be supported by XPress, but the results and the way it displays aren't reliable.*

columns

Number of columns in the text box (minimum 1; maximum 30).

gutter

Gutter width between columns (minimum 3 pt (but may be set to 0 pt if *columns* is 1); maximum 288 pt).

text inset

(see page 50) The size of the interior text margin on all four sides (minimum 0 pt; maximum 288 pt).

baseline offset

Offset of the first baseline in the text box (minimum 0 pt; maximum 648 pt). This parameter is used only when it is larger than both the type size and the text inset.

baseline minimum

Designates where the top of the text is; used by Xtags when the text inset value is what determines the placement of the first baseline. Use **C** (or **c**) to specify the first line's capital height, **A** (or **a**) to specify the first line's accent height, or "" (empty string or omitted parameter), to specify the first line's ascent height.

(For more information on the meaning of *baseline offset* and *baseline minimum*, see the anchored text box specifications documentation in the QuarkXPress manual.)

vertical alignment

Vertical alignment for text within the box; use **T** (or **t**) for top aligned text, **B** (or **b**) for bottom aligned, **C** (or **c**) for vertically centered, and **J** (or **j**) for vertically justified.

interparagraph max

Maximum amount of interparagraph leading, used only when *vert align* is **J** (minimum 0 pt; maximum 1080 pt; default 0 pt).

box name

Optional scripting-relative box name.

layer name

Unanchored boxes only. The layer on which the unanchored box will be created and placed. If the layer does not yet exist, it is created with default color and visibility. If the layer cannot be created, the box is placed on the currently selected (active) layer.

Example 7.1

Xtags Code:

```

@$:Xtags uses two tags to create and fill anchored text
boxes. <B>&tb<B> creates a text box and diverts the text
flow into it, and <B>&te<B> ends the text flow into the
box:¶
<*C*p(0, , , , 3, 1.25")>&tb(2.5", 1.2", A, 2, , 50, , ,
, , , 4, , , C)><z8f"Helvetica"i>This text will appear
inside of the anchored text box (framed 50% grey so you
can see it). Note how formatting changes made inside the
box don't persist once it ends, although those made in the
<BI>*p<BI> tag just before the <BI>&tb<BI> would remain in
effect if we didn't turn them off.<&te>¶
<*p$*J>The previous text style is now restored. This
example also shows how to place a specific amount of space
around an anchored text box: place it in its own paragraph,
with the appropriate settings given <I>before<I> the
<B>&tb<B> tag. This scheme works for text boxes which are
ascent-aligned (i.e., grow down from their text baseline),
but which need a large space after setting.¶
@$:Here's an example of a black box with white text:¶
<*R>&tb(1.5", 0.9", A, , , , K, 100, 3, , , 3, , ,
C)><*p(0,0,0)><z9B*CcW>This text will appear inside of
the anchored text box as knockout against its black
background.<&te>¶
<*L>Notice where text after the box ends up when you don't
set the space after in the box's paragraph. The next
paragraph starts on the next line and wraps around the box,
on the side with the most "room," just as if the anchored
box weren't really anchored, but manually placed where it
ends up.¶
<*p(.75", , , 12)*J>The other kind of in-paragraph box
alignment is baseline alignment, where the box is just
another (albeit tall) character on its text line. <&tb(1.
2", 10pt, B, , , , K, 70, 3, , , , B)><*p(0)*CcWz8B>I'm inside the
box.<&te> Then, the text in its paragraph continues on,
ignoring its settings completely.¶

```

Formatted Result:

Xtags uses two tags to create and fill anchored text boxes. **&tb** creates a text box and diverts the text flow into it, and **&te** ends the text flow into the box:

*This text will appear inside of the anchored text box (framed 50% grey so you can see it). Note how formatting changes made inside the box don't persist once it ends, although those made in the *p tag just before the &tb would remain in effect if we didn't turn them off.*

The previous text style is now restored. This example also shows how to place a specific amount of space around an anchored text box: place it in its own paragraph, with the appropriate settings given *before* the **&tb** tag. This scheme works for text boxes which are ascent-aligned (i.e., grow down from their text baseline), but which need a large space after setting.

Here's an example of a black box with white text:

Notice where text after the box ends up when you don't set the space after in the box's paragraph. The next paragraph starts on the next line and wraps around the box, on the side with the most "room," just as if the anchored box weren't really anchored, but manually placed where it ends up.

This text will appear inside of the anchored text box as knockout against its black background.

The other kind of in-paragraph box alignment is baseline alignment, where the box is just another (albeit tall) character on its text line. **I'm inside the box.** Then, the text in its paragraph continues on, ignoring its settings completely.

Example 7.2

Xtags Code:

```
@$:This tag creates a 2" x 0.25" unanchored text box at
the bottom of the page. <&tbu2(139.622,570,144,18,,,,n,
0.5,(,n),(,100),,,,,,3,,,,,)>@Example:An unanchored text
box!<&te> We've put a frame around it to make it easier to
find.
```

Formatted Result:

This tag creates a 2" x 0.25" unanchored text box at the bottom of the page. We've put a frame around it to make it easier to find.

An unanchored text box!

Tags for Picture Boxes

The **&pb** and **&pbu2** tags create and (optionally) fill an anchored or unanchored picture box. Their parameters are very similar—when a parameter is unique to an anchored or unanchored tag, we've underlined it. Parameters whose names are shown in **bold** also have more advanced forms which are covered in greater detail in later subsections. Parameters generally take their default values from the document's rectangular picture box tool, but those marked with a † have no tool defaults (see the parameter description for defaults assigned by Xtags).

The *anchored picture box tag* with its basic parameters has one of the following forms:

```
<&pb(width†, height†, anchored alignment†, frame width,
frame color, frame shade, frame style, background color,
background shade, text outset, placement†, picture scale
x, picture scale y, picture offset x, picture offset y,
picture angle, picture skew, picture path name†, picture
type†, box name†)>¶
```

The **&pb2** tag adds additional parameters (*box angle*, *box skew*, *flags*, and *item runaround*), and is otherwise identical:

```
<&pb2(width†, height†, box angle†, box skew†, flags†,
anchored alignment†, item runaround, frame width, frame
color, frame shade, frame style, background color,
background shade, text outset, placement†, picture scale
x, picture scale y, picture offset x, picture offset y,
picture angle, picture skew, picture path name†, picture
type†, box name†)>¶
```

The *unanchored picture box* tag with its basic parameters has this form:

```
<&pbu2(x†, y†, width†, height†, box angle†, box skew†,
flags†, item runaround, frame width, frame color, frame
shade, frame style, background color, background shade,
text outset, placement†, picture scale x, picture scale y,
picture offset x, picture offset y, picture angle, picture
skew, picture path name†, picture type†, box name†, layer
name†)>¶
```

Both the anchored and unanchored picture box tags have parameters that can be further expanded as sub-parameters. These include:

x: (*x offset*, *relative placement*, *relative box reference*, *relative box domain*)

width: (width, sizing specs, adjustment/margin, minimum)

height: (height, sizing specs, adjustment/margin, minimum, leading adjustment)

item runaround: (type, flags, index, noise, smoothness, threshold)

frame width: (frame width, corner diameter, corner type)

frame color: (frame color, frame gap color)

frame shade: (frame shade, frame gap shade, frame opacity, frame gap opacity)

background color: (background color, blend color, blend style, blend angle, start position, center position, end position)

background shade: (background shade, background opacity, blend shade, blend opacity)

text outset: (top, left, bottom, right)

picture path name: (path name, page number, cropping style)

Parameters for anchored and unanchored picture boxes

X
(see page 44) *Unanchored boxes only.* Horizontal page-relative position.

Y
Unanchored boxes only. Vertical page-relative position.

width
(see page 45) Width of the box (minimum is 2 pt; maximum 3456 pt; default 72 pt).

height
(see page 46) Height of the box (minimum is 2 pt; maximum 3456 pt; default 72 pt).

box angle
Pb2 and pbu2 tags only. Specifies the rotation in degrees about the bounding box midpoint, positive being counterclockwise and negative being clockwise (minimum: -360; maximum 360; default 0).

box skew
Pb2 and pbu2 tags only. The angle with which the box is skewed from vertical in degrees, with a negative value skewing the box to the left of vertical (minimum: -75; maximum: 75; default: 0).

flags

Pb2 and pbu2 tags only. An optional string consisting of one or more code or flag letters having the following meanings:

- ◆ **b** (or **B**) = **box** print suppression
- ◆ **p** (or **P**) = **picture** box content print suppression
- ◆ **k** (or **K**) = send the newly-created box behind all other boxes on the spread (as in “send to back”) (*Unanchored boxes only*)
- ◆ **l** (or **L**) = **lock** item
- ◆ **h** (or **H**) = flip box contents **h**orizontally
- ◆ **v** (or **V**) = flip box contents **v**ertically
- ◆ **u** (or **U**) = send box behind (under) its referenced box (The “k” flag will be ignored when both flags are given) (*Unanchored boxes only*)

anchored alignment

Anchored boxes only. Box alignment with respect to the text baseline for anchored boxes. You can optionally use **A** (or **a**) for ascent alignment (where the box grows down from the top of the ascender of the tallest character in its text line, if any) or use **B** (or **b**) for baseline alignment of the box (the default, where the created text box aligns with the baseline of its line as if it were a text character). For baseline offset, you must specify a value between -11” to 11.5”, in the form (**text align, offset**).

item runaround
(see page 48) *Pb2 and pbu2 tags only.* May be one of:

- | | QuarkXPress | InDesign |
|--|-------------|--|
| ◆ n (or N) = none | | no text wrap |
| ◆ i (or I) = item | | wrap around bounding box |
| ◆ j (or J) = item | | jump object |
| ◆ x (or X) = item | | jump to next column |
| ◆ a (or A) = auto image | | wrap around object shape: detect edges |
| ◆ b (or B) = picture bounds | | wrap around object shape: bounding box |
| ◆ c (or C) = alpha channel | | wrap around object shape: alpha channel |
| ◆ e (or E) = embedded path | | wrap around object shape: Photoshop path |
| ◆ s (or S) = same as clipping | | wrap around object shape: same as clipping |
| ◆ w (or W) = non-white areas | | wrap around object shape: detect edges |

frame width
(see page 48) Width of the box’s frame (minimum 0 pt (none); maximum 504 pt).

frame color

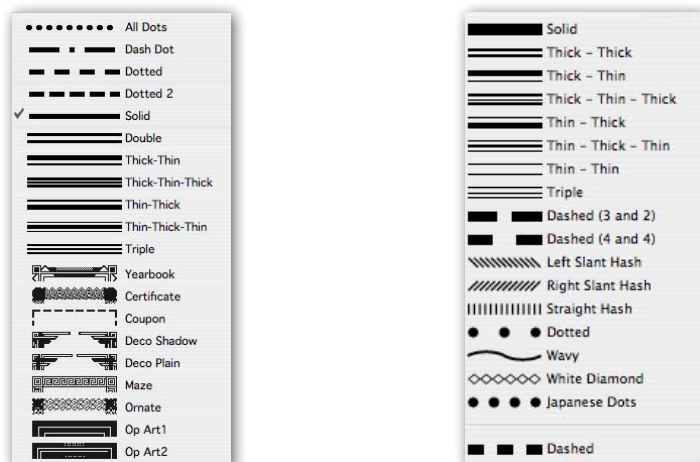
(see page 49) Color of the box's frame: either a quoted color name or one of the key characters as described under the <c> tag in Chapter 4.

frame shade

(see page 49) Shade of the box's frame, as a percentage (minimum 0%; maximum 100%).

frame style

Style of the box's frame, specified as the frame style name in quotes (e.g., "Solid"). Following is a list of the various applications and frame style names.



QuarkXPress frame names

InDesign frame names

InDesign Caveat: Xtags for InDesign accepts QuarkXPress algorithmic names as well as any InDesign frame names. QuarkXPress has several bitmapped styles (**Yearbook** through **Op Art2**) that are mapped to InDesign's **Solid** style, as is the **Dash Dot** style (since there's no InDesign equivalent style). English, French, and German-localized names are supported.

background color

(see page 50) Background color of the box, or **N** for no ("None") color.

background shade

(see page 50) Background shade of the box, as a percentage (minimum 0%; maximum 100%).

text outset

(see page 50) Text runaround exterior margin on all four sides of the box (minimum -288 pt; maximum 288 pt). Use negative values with caution: it

appears to be supported by XPress, but the results and the way it displays aren't reliable.

placement

Picture placement within the box. Use **C** (or **c**) for centered, **F** (or **f**) for expand-or-shrink-to-fit, **A** (or **a**) for expand-or-shrink-to-fit but maintaining the original aspect ratio, or **M** (or **m**) for manual (no special placement, the default setting). Any placement but **M** overrides some of the other parameters as needed to accomplish the requested placement (e.g., centering will set *picture offset x* and *y* as needed to center the picture, fitting of either sort will set *picture offset x*, *picture offset y*, *picture scale x*, and *picture scale y* as needed to make the picture fit the containing box, etc.).

picture scale x

Scaling percentage applied to the picture in the horizontal direction (minimum 10%; maximum 1000%).

picture scale y

Scaling percentage applied to the picture in the vertical direction (minimum 10%; maximum 1000%).

picture offset x

Horizontal offset of the picture from the left edge of the picture box (minimum is the negative of the native image width; maximum is the native image width).

picture offset y

Vertical offset of the picture from the top edge of the picture box (minimum is the negative of the native image height; maximum is the native image height).

picture angle

Angle in degrees of the picture within its (unrotated) picture box (minimum -360; maximum 360).

picture skew

Picture skew in degrees (minimum -75; maximum 75).

picture path name

(see the *picture handling section below*) Absolute or relative path to the picture (OR PDF).

picture type

A platform-independent key which represents the kind of picture file to be imported: **T** or **t** for TIFF, **J** or **j** for JPEG, **E** or **e** for EPS, **P** or **p** for PICT. Without this information, the **Update** button in the **Picture Usage.../Usage...** dialog isn't useful under Mac OS since it forces the picture to be updated be

the same type as the original, and if the picture is missing, QuarkXPress can't know that original type unless it's told. If you don't need to use this missing picture handling feature, you may omit this parameter.

box name

Optional scripting-relative box name.

layer name

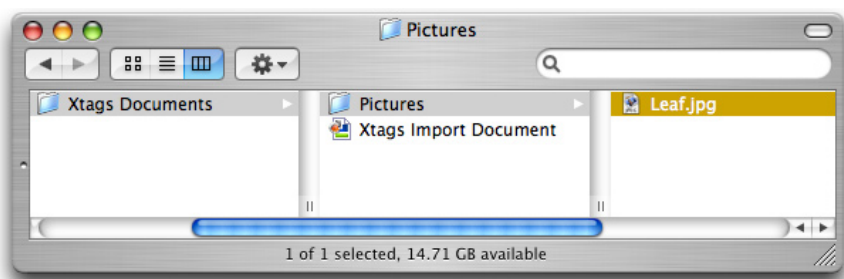
Unanchored boxes only. The layer on which the unanchored box will be created and placed. If the layer does not yet exist, it is created with default color and visibility. If the layer cannot be created, the box is placed on the currently selected (active) layer.

Picture handling

Path names are case-insensitive, but quite space sensitive; e.g., "Picture" is the same as "PICTURE", but distinct from "Pic ture". Here are some sample path names and their meanings:

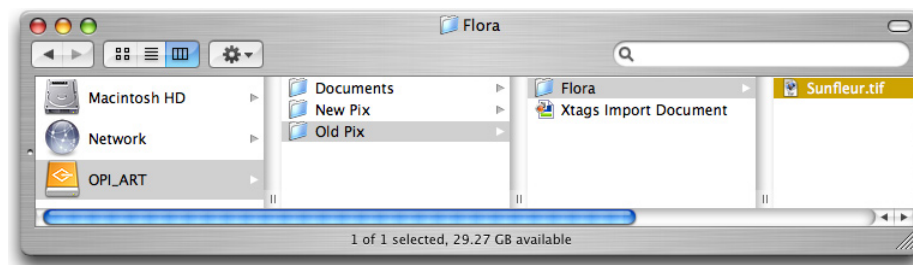
"Tree.TIF" is a picture file named Tree.TIF in the same folder as the current document.

Under Mac OS X, ":Pictures:Leaf.jpg", "Pictures/Leaf.jpg", or "./Pictures/Leaf.jpg" is a picture file named Leaf.jpg in the folder Pictures located either in the same folder as the current document or the folder containing the source file. Note the leading colon in the first case, or the dot/slash, which tell Xtags that this path name is relative. Under Windows, the corresponding picture file would be specified as "Pictures\\Leaf.jpg".



Mac OS X

Under Mac OS X "OPI_ART:Old Pix:Flora:Sunfleur.tif" is a picture file named Sunfleur.tif located on the disk named OPI_ART in the folder Flora, which is itself in the top-level folder named Old_Pix (or "/Volumes/OPI_ART/Old Pix/Flora/Sunfleur.tif" under Mac OS X). On a Windows system, the corresponding picture file would be something like "D:\\Old_Pix\\Flora\\Sunfleur.tif".



Mac OS X

If you are unsure about what path name to use for a particular file, then open a test document in QuarkXPress, noting the current folder. Import a graphic into a picture box, and then determine its full path name by using the **Utilities/Usage...** option.

If picture path name is omitted (which is perfectly legitimate if the picture is going to be manually imported later, or if the box is just for graphic effect), or the picture file cannot be found, then no picture is imported. There is no default picture path name.

When Xtags can't find a picture given by the picture path in any picture box creation tag, it leaves the given picture path as the internal name of the picture, for later potential automatic update by QuarkXPress (or from the **Usage...** dialog), or as a clue for someone to come along and import the picture manually. The preview for such missing pictures is a big red question mark.

Note that for this feature to work, at least under Mac OS, you must also use the appropriate picture type parameter in the picture box tag (discussed above).

InDesign Caveat: Missing pictures are not flagged with our missing picture proxy image (the red question mark) nor is the document primed with the missing picture's path.

Example 7.3

Xtags Code:

```
@Picture:<*J><*p(0,0,0,0,6.5,6.5,g,"U.S. English")><&pb(70
.655,48.085,a,0.5,(,),(,100),,,,(,5),m,45.2,45.2,,,,":W
indmill.jpg",,)>Anchored picture boxes work very much like
anchored text boxes. We're starting this paragraph with
one, and setting a 5 point runaround on the right side to
buffer the text from the picture. On the next line down
we've got a picture box on its own line. The tag creates
```


a picture box 5 inches wide and 5 inches high to hold the picture in the file `<@Filename>Windmill.jpg<@$>` (in the same folder as the current document). The picture is sized to fit the picture box and has a .5 point solid black frame.¶

```
@Picture:<*p(0,0,0,0,6.5,6.5,g,"U.S. English")><&pb((5"
,s),(5",s),,0.5,(,),,(100),,,,(),,m,,,,,"Windmill.
jpg",,)>¶
```

@Example Last:And here is some text following the picture.¶

Formatted Result:



Anchored picture boxes work very much like anchored text boxes. We're starting this paragraph with one, and setting a 5 point runaround on the right side to buffer the text from the picture. On the next line down we've got a picture box on its own line. The tag creates a picture box 5 inches wide and 5 inches high to hold the picture in the file `Windmill.jpg` (in the same folder as the current document). The picture is sized to fit the picture box and has a .5 point solid black frame.



And here is some text following the picture.

Conditional picture import

If the path name in a picture box creation tag begins with a question mark, no picture box is created (and no error is reported) if the file named by the rest of the path name doesn't exist. Note that the question mark must be inside the double quotes surrounding the file name.

Example 7.9

Xtags Code:

```
@$:<*p(.75",0,0,,.6",0)>Pictures can be imported
conditionally.¶
```

```
For example, this tag <&pb(1", 0.75",,,,,,,A,,,,,,
"?Container Ship.jpg")> will only create the given anchored
picture box if the file <@file name>Container Ship.jpg<@$p>
exists; otherwise, Xtags will silently proceed without
creating anything. That's what happens in this second
example <&pb(1", 2",,,,,,,A,,,,,, "?stone statue.jpg")>
where no picture box gets created.¶
```

Formatted Result:

Pictures can be imported conditionally.



For example, this tag will only create the given anchored picture box if the file `Container Ship.jpg` exists; otherwise, Xtags will silently proceed without creating anything. That's what happens in this second example where no picture box gets created.

Although these examples have used anchored picture boxes, the conditional picture importing features can be used with either anchored or unanchored picture boxes.

Extended picture path parameters

The *picture path name* parameter accepts a sub-list with the format:

```
(path name, page number, cropping style)
```

Note: *page number* and *cropping style* are silently ignored in QuarkXPress 6.x and for non-PDF image imports.

Path name is the quoted file system path denoting the picture file to be imported. Xtags supports the full set of picture types supported by the QuarkX-Press **Get Picture...** command. Either an absolute path or a path relative to the

current document's folder may be specified here. If importing tags from a file, the path may also be relative to the source file's folder, which will be searched last.

Note: For compatibility with the XPress Tags filter, the backslash character (\) is the escape character inside strings, and therefore must be doubled in path name strings to obtain a single backslash. For example, the Windows path name Pictures\Capital.tif must be specified as Pictures\\Capital.tif.

Page number is the number of the page to be imported. This parameter should be an integer in the range 1 ... *n*, where *n* is the number of the last page in the PDF. Values larger than *n* are clamped to *n*, while a value smaller than 1 will cause both the page and cropping parameters to be ignored. If not otherwise specified, Xtags will load the PDF's first page.

Cropping style is the type of the cropping method to be applied. It may be **0** (media), **1** (contents, the default), **2** (bleed), or **3** (trim).

Tags for None Boxes

The **&nb** and **&nbu2** tags create anchored or unanchored “none” boxes. None boxes do not have any content and consequently do not need a terminating tag. The **&nb** and **&nbu2** tag parameters are very similar—when a parameter is unique to an anchored or unanchored tag, we've underlined it. Parameters generally take their default values from the document's generic rectangle box tool (in QuarkXPress, there is no way to change these from the user interface), but those marked with a † have no tool defaults (see the parameter description for defaults assigned by Xtags).

The **anchored none box tag** with its basic parameters has one of the following forms:

```
<&nb(width†, height†, anchored alignment†, frame width,  
frame color, frame shade, frame style, background color,  
background shade, text outset, box name†)>
```

The **&nb2** tag adds additional parameters (*box angle*, *box skew*, *flags*, and *item runaround*), and is otherwise identical:

```
<&nb2(width†, height†, box angle†, box skew†, flags†,  
anchored alignment†, item runaround, frame width, frame  
color, frame shade, frame style, background color,  
background shade, text outset, box name†)>
```

The **unanchored none box tag** with its basic parameters has this form:

```
<&nbu2(x†, y†, width†, height†, box angle†, box skew†, flags  
†, item runaround, frame width, frame color, frame shade,
```

```
frame style, background color, background shade, text  
outset, box name†, layer name†)>
```

None box parameters are identical to the parameters for picture boxes, without picture-specific parameters, and will not be described here (*Descriptions can be found starting on page 37*). Parameters whose names are shown in **bold** also have more advanced, expanded forms which are covered in greater detail starting on page 44.

Tags for Lines

Xtags for QuarkXPress supports creating anchored and unanchored lines.

InDesign Caveat: Lines are not currently supported under InDesign.

Parameters for anchored line tags

The **&lb** tag creates an anchored line box:

```
<&lb(px†, py†, flags†, anchored alignment†, line width,  
line color, line shade, line style, text outset, line  
endcaps, box name†)>
```

Line box parameters generally take their default values from the document's line tool, but those marked with a † have no tool defaults (see the parameter description for defaults assigned by Xtags).

px

Denotes the horizontal component of the line, which is the distance from the line's initial point to its terminal point. A positive value “points” to the right side of the page, and a negative value will point towards the left side. If the **r** flag is specified (see the *flags* parameter, discussed below), then *px* denotes the line's polar angle.

py

Denotes the vertical component of the line, where a positive value moves toward the page bottom. If the **r** flag is specified (see *flags*), *py* denotes the line's length.

flags

An optional string consisting of one or more code letters with the following meanings: **b** (or **B**) = box print suppression, **l** (or **L**) = lock item, **o** (or **O**) = line is orthogonal, and **r** (or **R**) = line is a “ray” that extends at the specified angle (counter-clockwise from the horizontal axis) for the specified length, starting at the line box's anchor point in the text.³

³ Xtags-generated tags are written as rays, since this format is closest to QuarkXPress's internal representation.

anchored alignment

Box alignment with respect to the text baseline for anchored boxes. You can optionally use **A** (or **a**) for ascent alignment (where the box grows down from the top of the ascender of the tallest character in its text line, if any) or use **B** (or **b**) for baseline alignment of the box (the default, where the created text box aligns with the baseline of its line as if it were a text character).

line width

Width of the line (minimum 0 pt (hairline); maximum 864 pt).

line color

Color of the line. Optionally, this parameter may contain a sub-list specifying the line color and the line's gap color as (*line color, gap color*).

line shade

Shade of the line, as a percentage. Optionally, this parameter may contain a sub-list specifying the line shade and the line's gap shade as (*line shade, gap shade*).

line style

Style of the line, specified either as the line style name in quotes (e.g., "**Solid**") or as a line style index, an integer which runs from 1 to the number of line styles available.

text outset

Text runaround exterior margin on all sides of the line (minimum 0 pt; maximum 288 pt).

line endcaps

Arrowheads applied to the line, specified as the name in quotes, one of *Plain*, *Right*, *Left*, *Right Feathered*, *Left Feathered*, or *Double*.

box name

An optional scripting-relative box name.

Parameters for unanchored line tags

The **&lbu** tag creates an unanchored line box:

```
<&lbu(x†, y†, x2†, y2†, flags†, runaround, line width, line
color, line shade, line style, text outset, line endcaps,
box name†, layer name†)>
```

Line box parameters generally take their default values from the document's line tool, but those marked with a † have no tool defaults (see the parameter description for defaults assigned by Xtags).

x

Denotes the horizontal page-relative position for the new unanchored line's initial point. Note that the same relative placement options are supported as with the **x** parameter of the **&tbu2** and **&pbu2** tags.

y

Denotes the vertical page-relative position for the new unanchored line's initial point.

If **x** and **y** are present, then the line will be placed with its initial point at **x, y** on the current page. If **x,y** are absent, then the line is placed "more or less" where it would have been placed if it had been anchored with an ascent alignment.

x2

Denotes the horizontal position for the new line's terminal point. If the **r** flag is specified (see *flags*, below), then **x2** specifies the new line's polar angle with an origin of **x,y**.

y2

Denotes the vertical position for the new line's terminal point. If the **r** flag is specified, then **y2** specifies the new line's length.

flags

An optional string consisting of one or more code letters with the following meanings: **b** (or **B**) = **box** print suppression, **l** (or **L**) = **lock** item, **o** (or **O**) = line is **orthogonal**, **r** (or **R**) = line is a "**ray**" that extends at the specified angle (counter-clockwise from the horizontal axis) for the specified length starting at the line's initial point (specified by **x, y**), and **k** (or **K**) = send the newly-created box behind all other boxes on the spread (as in "send to back").⁴

runaround

Either **i** (or **I**) for item runaround or **n** (or **N**) for no runaround.

line width

Width of the line (minimum 0 pt (hairline); maximum 864 pt).

line color

Color of the line. Optionally, this parameter may contain a sub-list specifying the line color and the line's gap color as (*line color, gap color*).

line shade

Shade of the line, as a percentage. Optionally, this parameter may contain a sub-list specifying the line shade and the line's gap shade as (*line shade, gap shade*).

⁴ Xtags-generated tags are written as rays, since this format is closest to QuarkXPress's internal representation.

line style

Style of the line, specified either as the line style name in quotes (e.g., "Solid") or as a line style index, an integer which runs from 1 to the number of line styles available.

text outset

Text runaround exterior margin on all sides of the line (minimum 0 pt; maximum 288 pt).

line endcaps

Arrowheads applied to the line, specified as the name in quotes, one of *Plain*, *Right*, *Left*, *Right Feathered*, *Left Feathered*, or *Double*.

box name

An optional scripting-relative box name.

layer name

Unanchored lines only. The layer on which the unanchored line will be created and placed. If the layer does not yet exist, it is created with default color and visibility. If the layer cannot be created, the line is placed on the currently selected (active) layer.

Example 7.4

The following code creates an unanchored, horizontal 2" black rule, 1 point in thickness.

Xtags Code:

```
<&1bu(59.327,391.573,0,144,or,,1,,,,,)>¶
```

Formatted Result:

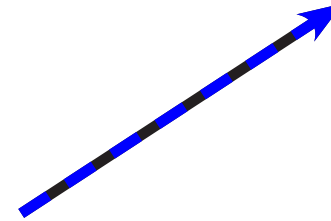


The following code creates an anchored 2" dotted blue (70%) and black (30%) rule, 4 pts in thickness, with an arrowhead. It's not on a 90 degree angle, so Xtags refers to this as a ray (*r*).

Xtags Code:

```
<&1b(33.134,144,r,,4,("Blue",K),(70,30),"Dotted",1,"Right",)>¶
```

Formatted Result:



Expanded Box Creation Parameters

Several of the parameters to box creation tags may be specified more fully as a sub-list of parameters, giving you access to the full range of settings for a particular parameter, to achieve certain effects. These include:

- ◆ the *x* parameter of any unanchored box;
- ◆ the *width* parameter of any box;
- ◆ the *height* parameter of any box;
- ◆ the *item runaround* parameter of any picture box;
- ◆ the *frame width* parameter of any box;
- ◆ the *frame color* parameter of any box;
- ◆ the *frame shade* parameter of any box;
- ◆ the *background color* parameter of any box;
- ◆ the *background shade* parameter of any box;
- ◆ the *text outset* parameter of any box;
- ◆ the *text inset* parameter of any text box;
- ◆ the *picture path name* parameter of any picture box.

We will consider the advanced forms of each of these parameters in the subsections that follow, according to their function.

Absolute and Relative Box Placement

Xtags lets you place unanchored boxes at an absolute position on a page, relative to the current insertion point, or relative to other boxes you've created (or to the pasteboard).

But first, note that if any kind of placement causes the bottom of the box to protrude past the bottom of the spread, then the box is moved up so its bottom

coincides with the spread bottom. Independently, if the right edge of the box protrudes past the right edge of the spread, then the box is moved left so its right edge coincides with the right edge of the spread.

If *x* and *y* are present in their simple form in an unanchored box creation tag, then the box is placed absolutely with its top left corner at position (*x*, *y*) on the current page. **Be aware that the notion of a “current page” is fraught with complexity; it’s best to use these position parameters only if you’re building one page or are fully aware of page creation.**

If both *x* and *y* are absent, then the box is placed “more or less” where it would have been placed if it had been an anchored box with an ascent alignment (i.e., the top of the box will be placed at the top left of the current insertion point at the time of this tag).

One caveat for the `&tbu2/&pbu2/&nbu2` tag (due to a QuarkXPress limitation): If you’re going to use back-to-back unanchored box tags with no intervening text, you should first insert enough pages to hold all the unanchored text boxes (taking care to link these pages into the main text flow) and then do the Xtags import. Otherwise, after two pages, QuarkXPress will overflow the automatic text box (even though it should keep creating pages, logically speaking) and you’ll get an Xtags error (**Can’t place box: text overflow**) for each unanchored box from that point on.

Finally, if the *x* coordinate is specified as a sub-list, it indicates the box’s placement with respect to another box. It takes the form of the following sub-list:

(x offset, relative placement, relative box reference, relative box domain)

x offset is the horizontal offset from the referenced box, rather than an absolute coordinate.

Relative placement specifies the type of relative placement: **B** or **P** for box placement on the pasteboard (relative to the top left of the pasteboard), or **TL** (top left), **TR** (top right), **BL** (bottom left) or **BR** (bottom right) for placement relative to another box, the relatively-referenced box. The actual placement is offset horizontally by the value of *x offset*, and vertically by the value of the *y* coordinate (which thus effectively becomes an offset).

Relative box reference selects a previously-created box to be used as a reference when placing this new box. The box can be chosen explicitly by name, or, an integer from 1 to 100 can be given, naming the *n*th unanchored box most recently created by Xtags.

For example, the *x* coordinate sub-list (**3pt, BL, 2**) specifies that the top left corner of the about-to-be-created box is to be placed 3 points to the right of the bottom left corner of the second-most-recently created box.

A relative box reference of **0** (zero) refers to the most-recently-created anchored box in the Xtags input (if any). This means you can create an anchored box, then create other, unanchored boxes relative to it.⁵

If a named box is given as a reference, but is not found, the box creation fails and an error is generated. If the name is prefaced with a question mark, the creation is considered conditional: if the named box is not found then no box is created and no error is reported.

Relative box domain tells Xtags where to search for a named box. Valid options are:

- ◆ **S** = Search the current spread (the default).
- ◆ **P** = Limit the search to the current page.
- ◆ **D** = Search the document’s entire layout.

Example: `<&tbu2((0,TL,"background",D),0,(1,R,0),(1,R,0),...)>` searches the document for a box named “background” and creates a same-sized text box on top of it.

Note that you can name a box on a master page and then position boxes on each document page relative to where the named master page box lies. This could be handy when tagged elements’ placements need to be determined by the layout rather than the tags. For example, something like: `<&tbu2((0,TL,"?left-page-footer",P),0,(1,R,0),(1,R,0),...)>` might be used to place a left-page footer only on left pages. Since no box named “left-page-footer” would exist on the right page master, the tag would silently fail on right-side pages.

Automatic Box Resizing

Expanded width and height parameters allow you to more precisely specify box and/or contents resizing, including separate horizontal (width) and vertical (height) resizing specifications.

Box width parameter

A *width* parameter can take the form of the following sub-list:

(width, sizing specs, adjustment/margin, minimum)

⁵ You should probably use a runaround of “None” for any such unanchored boxes you create, in order to avoid disturbing the location of the anchored box you’re referencing.

where *width* is the initial width of the box (must be greater than 0), and *sizing specs* contains one or more of the following code letters:

- S** or **?** shrink-to-fit-contents sizing (shrinks the box width to fit the contents);
- R** relative sizing to any relative placement box (see the *x* coordinate specifier);
- F** fit-to-box-width picture content sizing (shrinks or expands the content picture to fit the box width).

Adjustment/margin is either the amount (positive or negative) by which to adjust the relative width (in the case of relative sizing), or the margin (positive or negative) for sizing the box to fit its contents in all other cases. In the latter form, a positive margin adds width inside the box to the natural width of the contents, and a negative margin (which is only valid for picture boxes) subtracts width inside the box from the natural width of the contents. This parameter defaults to 0.

Minimum is the minimum width permitted by either type of resizing, and defaults to 2 points. (The maximum width is the initial width of the box, specified by *width*; boxes are never grown as a result of automatic resizing.)

For example, the width parameter sub-list (**2.0" , S , -2mm , 1.5"**) specifies a picture box created with an initial width of 2 inches, and the width will be resized to fit its contents. The box width will be shrunk to 2mm smaller than the natural width of its contents, but no smaller than a minimum width of 1.5".

Box height parameter

Similarly, a *height* parameter can take the form of the following sub-list:

(*height, sizing specs, adjustment/margin, minimum, leading adjustment*)

Height is the initial height of the box (as well as its maximum height). This value must be greater than 0.

Sizing specs contains one or more of the following codes:

- S** or **?** shrink-to-fit-contents sizing (shrinks the box height to fit the contents);
- R** relative sizing to any relative placement box (see the *x* coordinate specifier);
- L** force the containing paragraph's leading to the final box height (the latter applies to anchored boxes only);

- F** fit-to-box-height picture sizing (shrinks or expands the content picture to fit the box height). Note that this option cannot be used at the same time as fit-to-box-width;

- B** force space above and below.

Adjustment/margin is either the amount (positive or negative) by which to adjust the relative height (in the case of relative sizing), or the margin (positive or negative) for sizing the box to fit its contents in all other cases. In the latter form, a positive margin adds height inside the box to the natural height of the contents, and a negative margin (which is only valid for picture boxes) subtracts height inside the box from the natural height of the contents. This parameter defaults to 0.

Minimum is the minimum height permitted by either type of resizing, and defaults to 2 points. (The maximum height is the initial height of the box, specified by *height*; as before, boxes are never grown as a result of automatic resizing.)

Leading adjustment is the amount (positive or negative) to be added to the box height for anchored boxes in order to set the leading for the containing paragraph. This parameter is used only when *sizing specs* includes the **L** code, and it defaults to 0. If *sizing specs* includes the **B** code, this parameter specifies the amount of additional space below the anchored box's containing paragraph (it adjusts the containing paragraph's space below rather than leading).

For example, the *height* parameter sub-list (**3.0" , SL , 1mm , 1.0" , 3pt**) specifies that the height of an anchored box be initially set to 3". The box will be shrunk to fit the natural height of its contents plus 1mm or to a minimum height of 1". The paragraph leading of the paragraph in which the box is located will be set to the final box height plus 3 points.

Shrink-to-fit text boxes

Utilizing the automatic box sizing feature, you can make text boxes shrink to fit their content. The box is initially created at the given size and the containing text processed; then the box is examined to see how much space is actually needed. Any shrink-to-fit-margins are added to the sides (along with any frame width and text inset), and the box is resized appropriately. Shrinking-to-fit will fail for a text box that is empty or overflowing.

Text boxes can shrink-to-fit in either or both dimensions. Vertical sizing takes precedence over horizontal. The box will be reduced until any further decrease in size would cause the text inside to overflow.

Note: The box is never expanded, so it must be created at the maximum desired size. Thus, you must be careful that you never create a box height or width that exceeds the text flow's column height or width, or the story will overflow.


Example 7.5

Xtags Code:

```
@Example:<*p(,,,,.15",0)>This tag sequence <&tb(1", 3"?1pt,
, , , ,K,15 , , , , 4, , , C)><B><*p(0,0,0)><*C><z8>text
in the box<&te>will create a box 1" wide and initially
3" high, put the text in the box, and then shrink it
vertically to fit, leaving 1 point of margin above and
below the text.¶
```

```
@Example Last:Similarly, this tag sequence <&tbu2(382,358,
1", (3",s),90,,,,,(,), (,),K,15,6,,,4,,,c,,,)>@$:<*C><*p(0,
0,0,13,0,0,g,"U.S. English")><B><z8>I'm inside...<&te>will
create an unanchored box 1" wide and initially 3" high,
rotate the box by 90 degrees, put the text in the box,
shrink it to fit vertically, put a six-point runaround on
it, and place it in the left margin of this column.¶
```

Formatted Result:

This tag sequence  will create a box 1" wide and initially 3" high, put the text in the box, and then shrink it vertically to fit, leaving 1 point of margin above and below the text.

I'm inside...

Similarly, this tag sequence will create an unanchored box 1" wide and initially 3" high, rotate the box by 90 degrees, put the text in the box, shrink it to fit vertically, put a six-point runaround on it, and place it in the left margin of this column.

Shrink-to-fit picture boxes

Utilizing the automatic box sizing feature, you can make picture boxes shrink to fit their content. The box is initially created at the given size, the picture is imported, and the box is shrunk to fit the picture's appropriate natural dimension (given the positive or negative *picture scale x* or *picture scale y* values and the given margin on both sides of a given dimension). This shrinking-to-fit is performed for each dimension independently. The final step is picture placement.

Note: The box is never expanded, so it must be created at the maximum desired size. Thus, you must be careful that you never create a box height or width that exceeds the text flow's column height or width, or the story will overflow.

Shrinking-to-fit for a picture box will fail if the picture box is empty, or if the "shrunk" size would actually expand the box.

Additionally, there are both "fit-to-height" and "fit-to-width" features (F in the sizing specs) for picture boxes of any flavor that can be combined with the shrink-to-fit feature in the other dimension. For example, the anchored picture box creation tag `<&pb((2", s), (2", f), ...)>` creates an anchored picture box initially two inches square, imports the given picture, and then sets the *x* and *y* scale of the imported picture such that the picture exactly fits the box vertically (i.e., it fits the picture to the box) and finally shrinks the box width to fit horizontally at that scale (fits the box to the picture). The two sizing specs can be swapped to achieve the inverse effect, where the picture is first sized to fit the box horizontally, and then the box is shrunk to fit vertically.

Example 7.6

Xtags Code:

```
@$:<*p(.75",0,0,,0.8",12pt)>For example, this tag
<&pb((1",s,-1mm), (2",s,-1mm),, 0.5pt, ,,,,,,C,,,,,,,"child
with moms shoes.jpg",,)> will create an anchored box of
1" by 2", import the same picture file, shrink the box to
fit its contents but with a 1mm negative margin or outset
(meaning the box will be 1mm smaller on all four sides than
the image) and then center the picture in the resulting
box.¶
```

Formatted Result:



For example, this tag will create an anchored box of 1" by 2", import the same picture file, shrink the box to fit its contents but with a 1mm negative margin or outset (meaning the box will be 1mm smaller on all four sides than the image) and then center the picture in the resulting box.

Picture Box Runaround Types (unanchored only)

Item runaround

the *item runaround* parameter of the **pb2** and **pbu2** tags can expand into the following sub-list:

(*type, flags, index, noise, smoothness, threshold, [side]*)

Type may be one of:

QuarkXPress	InDesign
◆ n (or N) = none	no text wrap
◆ i (or I) = item	wrap around bounding box
◆ j (or J) = item	jump object
◆ x (or X) = item	jump to next column
◆ a (or A) = auto image	wrap around object shape: detect edges
◆ b (or B) = picture bounds	wrap around object shape: bounding box
◆ c (or C) = alpha channel	wrap around object shape: alpha channel
◆ e (or E) = embedded path	wrap around object shape: Photoshop path
◆ s (or S) = same as clipping	wrap around object shape: same as clipping
◆ w (or W) = non-white areas	wrap around object shape: detect edges

Flags may specify any (or none) of the following: **I** (invert), **E** (include inner edges), **O** (let path extend outside the box).

Index specifies the embedded path or alpha channel when one of those types is selected (minimum: 0; maximum 255).

Noise specifies the tolerance settings for auto image, alpha channel, and non-white runaround (minimum: 0; maximum 288pt).

smoothness specifies the tolerance settings for auto image, alpha channel, and non-white runaround (minimum: 0; maximum 100pt).

threshold specifies the tolerance settings for auto image, alpha channel, and non-white runaround (minimum: 0; maximum 100%).

side (*InDesign CS3 only*⁶) specifies InDesign's **Wrap To** setting, which lets the user choose which side or sides of the object text will flow, and may be one of: **R** (right side), **L** (left side), **B** (both right & left sides), **T** (side towards spine), **A** (side away from spine), **G** (largest area, the default).

Box Frame Specifications

Frame width

For rectangular boxes, the *frame width* parameter takes the form of the following sub-list (defaults are taken from the tool preferences):

(*frame width, corner diameter, corner type*)

For oval boxes, the form of the sub-list is:

(*frame width, frame shape*)

Frame width is the width of the frame.

Corner diameter is the diameter of any non-straight corner.

InDesign Caveat: Corner diameters are not limited by the height and width of the box; i.e., InDesign does not keep corners round like QuarkXPress does.

Corner type is **v** for convex, **c** for concave, or **s** for straight. Default is convex.

Frame shape is **O** (or **o**) = for oval box.

For example, the *frame width* specification (**4, .25", c**) in a box creation tag will create a frame with a width of 4 points having concave corners with a diameter of one quarter-inch.

⁶ InDesign CS2 does not have the "Wrap To" setting; it always uses "both right & left sides." QuarkXPress always uses "largest area."

Frame color

The *frame color* parameter of a box-creation tag can take the form of the following sub-list:

```
(frame color, frame gap color)
```

This gives you control over both the foreground and background colors in a frame.

Frame color may be specified as **n** to select **no** (“None”) **color** (as in other color cases).

Frame shade

The *frame shade* parameter in a box-creation tag can take the form of the following sub-list:

```
(frame shade, frame gap shade, frame opacity†, frame gap opacity†)
```

Opacity sub-parameters (marked with †) only work in QuarkXPress 7.x, and are ignored in both QuarkXPress 6.x and InDesign.

Frame shade and *frame gap shade* are the frame’s foreground and background colors, respectively, given as a percentage (minimum 0%; maximum 100%).

Frame opacity and *frame gap opacity* (QuarkXPress 7.x only) are the frame’s foreground and background opacities, respectively, given as a percentage (minimum 0%; maximum 100%).

Example 7.7

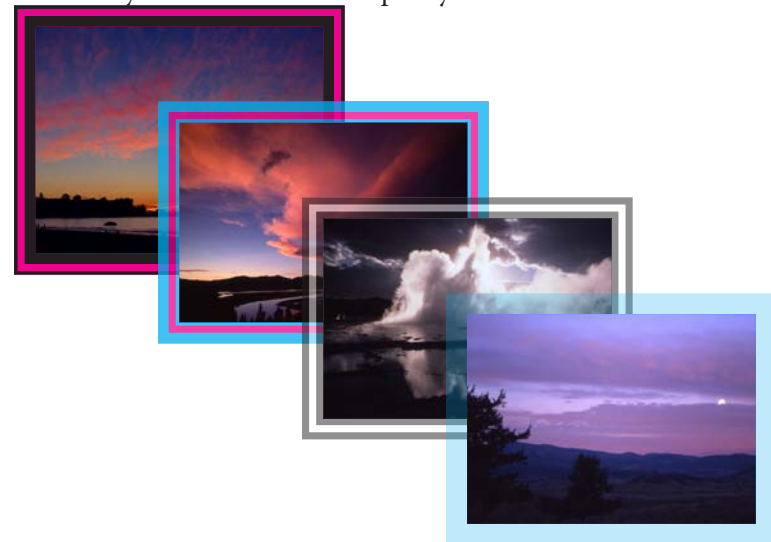
Xtags Code:

```
@$:Our example contains four cascading picture boxes, each
with 8 pt. frames that have the following properties:¶
1: Black and magenta, both totally opaque.¶
2: Cyan and magenta, both 75% opaque.¶
3: Black (50% opacity) and white (20% opacity) frame.¶
4: Cyan frame with 25% opacity.¶
<&pbu2(.5",2", (1.5",F), (1.5",S),,,,,8, (K,M), (,100,,), "Thi
n-Thick",,,,,m,,,,,"sunset 1.jpg")>¶
<&pbu2(1.25",2.5", (1.5",F), (1.5",S),,,,,8, (C,M), (,100,75,7
5), "Thick-Thin",,,,,m,,,,,"sunset 2.jpg")>¶
<&pbu2(2",3", (1.5",F), (1.5",S),,,,,8, (K,W), (,100,50,20), "D
ouble",,,,,m,,,,,"sunset 3.jpg")>¶
<&pbu2(2.75",3.5", (1.5",F), (1.5",S),,,,,8,C, (,100,25),,,,,
,m,,,,,"sunset 4.jpg")>¶
```

Formatted Result:

Our example contains four cascading picture boxes, each with 8 pt. frames that have the following properties:

- 1: Black and magenta, both totally opaque.
- 2: Cyan and magenta, both 75% opaque.
- 3: Black (50% opacity) and white (20% opacity) frame.
- 4: Cyan frame with 25% opacity.



Box Background Opacity and Blends

Xtags for QuarkXPress 7.x supports opacity and blends on any box. These features are controlled by providing additional sub-parameters instead of simple *background color* or *background shade* parameters.

InDesign Caveat: Box opacities and blends are not currently available in Xtags for InDesign, and are silently ignored.

Background color

background color can take an expanded sub-parameter list, allowing you to create a background blend on the box:

(background color, blend color, blend style, blend angle)

Background color is the first color component of the blend (**N** for “None”). The default is taken from the tool preferences.

Blend color is the second color component of the blend (**N** for “None”). “White” is the default.

Blend style may be one of: **S** (or “Solid”), **L** (or “Linear”), **M** (or “Mid-Linear”), **R** (or “Rectangular”), **D** (or “Diamond”), **C** (or “Circular”), **F** (or “Full Circular”). If not otherwise specified, a blend’s style defaults to linear.

Blend angle optionally rotates the blend (default is **0**).

Background shade

The *background shade* parameter has been augmented to modify both the blend shade and opacity fields:

(background shade, background opacity, blend shade, blend opacity)

background shade and *background opacity* should be specified as a percentage. (minimum 0%; maximum 100%; default taken from tool’s setting).

blend shade and *blend opacity* should be given as a percentage. (minimum 0%; maximum 100%; default 100%).

Example 7.8

Xtags Code:

```
<tbu2(44.558,83.8,216,47,,,-40,v,n,,,,,n)>@$:<*C><y150z30>
Linear Blend<te>¶
```

```
<tbu2(.5",1",3",.9",,,,n,,,,(K,M,L,90),(,95,,80),,,,,,
,,)>@$:<*C><z30>Linear Blend<te>¶
```

Formatted Result:



Text Insets and Outsets

The *text inset* and *text outset* parameters enable you to specify the four amounts individually. Both of these sub-lists take the following form:

(top, left, bottom, right)

Any omitted value is replaced with the default for the appropriate tool.

Grouping Tags

Grouping Unanchored Boxes

The **&g** tag is used to group unanchored recently-created boxes of any type, and takes the form:

```
<&g(n1, n2, ... nm)>
```

where each n_i is a relative reference to the n_i th most-recently-created box (for example, **1** designates the most-recently-created unanchored box, **2** designates the second-most-recently-created box, and so on). You can reference up to 100 most-recently-created boxes. Groups may be nested, so an n_i may refer to a group box (recently created using the **&g** tag) as well as to a text, picture or line box.

For example, the following tag would group the fourth and seventh boxes most recently created with Xtags:

```
<&g(4, 7)>
```

Note that the **&g** tag itself creates a group box (containing the group) which may be referenced as a recently-created box, and must be taken into account in future relative box reference number computations.

Grouped items, including groups of grouped items—but not a set of multiple-selected objects that are not grouped—may be input and output with the normal **Copy Xtags Text** and **Paste Xtags Text** (**Copy with Xtags** and **Paste with Xtags** in QuarkXpress 7.x) items on the **Edit** menu (the latter in item tool mode only).

Example 7.10

Xtags Code:

```
@Example:Here is an example of a picture and caption. The
6"x6" picture box is created first.<&pbu2(408,454.971,
(6",S),(6",S),,,,,4,(n),(,100),"Thick-Thin",n,,(6,0
,6,12),m,27.344,27.344,,,,":red toy car.jpg",,,)>¶
The picture box is shrunk to fit the picture both
horizontally and vertically. The box has a a 4 point, 100%
black frame set to a thick-thin stroke. We've specified a
top text outset of 6 points; left text outset of 0 points;
bottom outset of 6 points, and a right outset of 12
points.¶
After the picture box is created, a small text box is
created just below it using relative positioning and
filled with a caption<&tbu2((0,BL,1), 0.05", 1.2", 14pt,
,,,,,,(0,0,12pt,109),,,0.5pt)>*p(0,0,0,,0,0)>*C<z
8I>A classic retro toy car.<&te>. The text outset on the
text box is 12 points on the bottom and 109 points on the
right.¶
Finally, we're going to <&g(2, 1)>group the two boxes
together.¶
```

Formatted Result:

Here is an example of a picture and caption. The 6"x6" picture box is created first.



A classic retro toy car.

The picture box is shrunk to fit the picture both horizontally and vertically. The box has a a 4 point, 100% black frame set to a thick-thin stroke. We've specified a top text outset of 6 points; left text outset of 0 points; bottom outset of 6 points, and a right outset of 12 points.

After the picture box is created, a small text box is created just below it using relative position-

ing and filled with a caption. The text outset on the text box is 12 points on the bottom and 109 points on the right.

Finally, we're going to group the two boxes together.

Set/Clear Relative Origin Tags

The "set relative origin" tag⁷ has the form:

```
<&o(x, y, flags)>
```

and sets the relative origin for subsequent box creation tags to (x, y), with cumulative transformation (i.e., multiple set relative origin tags are additive) unless flags contains **s** for "sticky", which means this new origin stays in effect until cleared, regardless of following set relative origin tags.

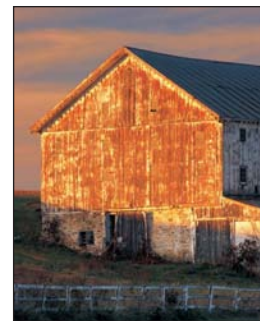
The "clear relative origin" tag can take the form of either <&o\$> or <&o0> and clears any relative origin in effect.

Example 7.11

Xtags Code:

```
<&o(52.305,324.335)><&pbu2(0,0,98.439,118.497,,,,n,0.5,(,
,(,100),,n,,m,22.658,22.717,,,,,"Old Barn.jpg",,,)>&tbu2
(103.536,96.542,96.753,24.955,,,,n,,(n),(,100),,n,,,,,
,b,,)>@$:<*L><*p(0,0,0,11,0,13.032,g,"U.S. English")><I>A
beautiful old barn.<I><&te><&g(2,1)>¶
```

Formatted Result:



A beautiful old barn.

⁷ You won't normally need this tag, but Xtags uses it on output to set an origin for a group of boxes. This way, the whole group can be easily relocated by changing the origin given by this tag.

Chapter 8

Creating Tables

Xtags Pro provides several powerful tags for table building, as described in this chapter. (These features are not available in the normal Xtags product, though you can upgrade to the Pro version from the normal version.)

InDesign Caveat: Tables are currently unsupported under InDesign, although you can create them indirectly using InDesign Tags within the `<&it""..."">` tag.

Tags for Tables

The table start (`&ts` and `&tsu`) tags create anchored or unanchored table boxes, respectively. They must be used in conjunction with table row (`&trs`) and table cell (`&tcs`) tags. All normal text inside a table tag but outside of cell tags is ignored. This makes table tag formatting easier because white space can be placed between tags without affecting the document.

Parameters for table start and table cell tags are similar to other box creation parameters. When a parameter is unique to an anchored or unanchored tag, we've underlined it. Parameters whose names are shown in **bold** have optional, advanced sub-parameters. Parameters generally take their default values from the document's table tool, but those marked with a † have no tool defaults (see the parameter description for defaults assigned by Xtags). Parameters within [brackets] only exist in `<v7.00>` tags—for other input source levels, these parameters should be omitted altogether.

The *anchored table box tag* with its basic parameters has this form:

```
<&ts(width†, height†, number of columns, number of rows,
column widths, flags, anchored alignment†, frame width,
frame color, frame shade, frame style, background color,
background shade, text outset, gridline width, gridline
style, gridline color, gridline shade, [gridline opacity],
gridgap color, gridgap shade, [gridgap opacity], tab order,
link order, box name†)>...rows...<&tse>
```

The *unanchored table box tag* with its basic parameters has this form:

```
<&tsu(x†, y†, width†, height†, number of columns, number of
rows, column widths, table angle†, flags, item runaround†,
frame width, frame color, frame shade, frame style,
background color, background shade, text outset, gridline
width, gridline style, gridline color, gridline shade,
[gridline opacity], gridgap color, gridgap shade, [gridgap
opacity], tab order, link order, box name†, layer name†)>...
rows...<&tse>
```

All character and paragraph settings that are in effect when a table is created are saved by Xtags and are restored when the `&tse` (table end) tag is encountered.

Both the anchored and unanchored table box tags have parameters that can be further expanded as more complex sub-parameters. These include:

```
x: (x offset, relative placement, relative box reference,
relative box domain)

width: (width 1, width 2, width 3, ...)

height: (height, sizing specs, adjustment/margin, minimum,
leading adjustment)

column widths: (column 1 width, column 2 width, column 3
width, ...)

frame width: (frame width, corner diameter, corner type)

frame color: (frame color, frame gap color)

frame shade: (frame shade, frame gap shade, frame opacity,
frame gap opacity)

background color: (background color, blend color, blend
style, blend angle, start position, center position, end
position)

background shade: (background shade, background opacity,
blend shade, blend opacity)

text outset, text inset: (top, left, bottom, right)

gridline width, gridline style, gridline color, gridline
shade, gridline opacity, gridgap color, gridgap shade, and
gridgap opacity: (top, left, bottom, right, all interior
horizontal, all interior vertical)
```

Parameters for anchored and unanchored table boxes

x (see page 44) *Unanchored tables only.* Horizontal page-relative position.

y *Unanchored tables only.* Vertical page-relative position.

width, height

The **width** and/or **height** parameters for the table can be omitted, in which case Xtags will default to the “natural” width and/or height of the table. Each parameter can either be specified as a measurement (useful for fixing the table to a pre-defined size), or a percentage of some surrounding element's size. Element percentages are specified by:

- ◆ **C%** = **C**olumn of text box at current insertion point
- ◆ **B%** = **B**OX
- ◆ **P%** = **P**age
- ◆ **S%** = **S**pread

where % is the scaling factor; minimum 0.01%; maximum 800%; default 100%.

The **width** parameter can be further expanded into sub-parameters, each taking the same form as the top-level **width** parameter just described:

width: (*width 1, width 2, width 3, ...*)

If multiple widths are specified, Xtags will evaluate each one and “snap” to the width that is closest to, but not less than, a “natural” size. A natural size is one in which each column is just wide enough for its contents to be displayed unbroken (unhyphenated and not overset, in the case of text).

The **height** parameter can also be expressed as: *height^break height*. The (optional) *break height* component tells Xtags how tall the table can get before breaking (it will default to the maximum possible size if omitted). QuarkXPress only supports breaking for unanchored tables.

The height parameter can also be expressed as a list of sub-parameters. These are the same as the expanded height sub-parameters for all other types of boxes, with the exception of the first sub-parameter, which behaves as described above:

height: (*height, sizing specs, adjustment/margin, minimum, leading adjustment*) (See page 49).

number of columns, number of rows

Hints for Xtags that will speed up the table building process. If these parameters are omitted, then the rows/columns will be dynamically created, based

on the given table row and cell tags. Tables will still be built correctly, but less efficiently. The maximum number of columns is currently limited to 64.

column widths

Column widths is an optional sub-list of column widths (in any units). If they're given and don't add up to an (optional) given overall table width, they're all scaled appropriately (and relatively) to match. Missing column widths default to .5" (36pt).

The *column widths* parameter may also be used to specify each column's alignment. A | (vertical bar) character after any width value signals that the contents of the cells in that column should be aligned.

An optional alignment position may follow the | (vertical bar), and is specified as:

- ◆ a fixed left-indent position (e.g. **12pt**)
- ◆ a percentage of the column's width (as in *CALS*, e.g. **50%**)
- ◆ one of **L**, **C**, or **R** = **L**eft-aligned, **C**entered (the default, which is really just 50%), or **R**ight-aligned.

By default, the content will be aligned on the first occurrence of . (decimal point). This behavior can be changed by adding an optional single-quoted alignment character after the alignment position (*'character'*). Commas and single-quote characters must be “escaped” with a backslash in order to be used: '\, ' and '\ ' .

Let's look at the column widths from the following example in greater detail:

Column 1: (**1" | , | 75% , 2" | 1.25" '\, '**)

Width is 1". Since no alignment position is given after the | (vertical bar), the cell's contents will be shifted so that the first period is centered. (The bar causes the contents to be aligned. Centered and period are the defaults.)

Column 2: (**1" | , | 75% , 2" | 1.25" '\, '**)

Width is not given, so Xtags will automatically assign a reasonable value. The contents will be shifted so that the first period is placed 75% of the way across the cell. Because of reduced width constraints, the contents appear to be right-aligned.

Column 3: (1" | , | 75%, 2" | 1.25" '\, ')

Width is fixed at 2". The contents of the cells in that column will be shifted so that the first comma will be aligned 1.25" from the left side of the column.

Example 8.1

Xtags Code:

```
<&tsu(1",1", , ,3,3,(1" | , | 75%,2" | 1.25" '\, '))>¶
<&trs(,36)><&tcs>123.45<&tcs>9.95<&tcs>the,quick<&tre>¶
<&trs(,36)><&tcs>0.1<&tcs>19.95<&tcs>brown,fox<&tre>¶
<&trs(,54)><&tcs>12345.99<&tcs>1.19¶
2.45<&tcs>jumps,over¶
the,lazy dog<&tre>¶
<&tse>¶
```

Formatted Result:

123.45	9.95	the,quick
0.1	19.95	brown,fox
12345.99	1.19 2.45	jumps,over the,lazy dog

table angle

Unanchored tables only. Specifies the rotation in degrees about the bounding box midpoint, positive being counterclockwise and negative being clockwise (minimum: -360; maximum 360; default 0).

flags

An optional string consisting of one or more code or flag letters having the following meanings:

- ◆ **b** (or **B**) = **box** print suppression
- ◆ **c** (or **C**) = set table's auto-fit **c**olumns option
- ◆ **k** (or **K**) = send the newly-created box behind all other boxes on the spread (as in "send to **back**")

- ◆ **l** (or **L**) = **l**ock newly-created table
- ◆ **m** (or **M**) = **m**aintain geometry (changing the size of a row/column doesn't resize the table)
- ◆ **r** (or **R**) = set table's auto-fit **r**ows option

anchored alignment

Anchored tables only. Table alignment with respect to the text baseline for anchored tables. You can optionally use **A** (or **a**) for ascent alignment (where the table grows down from the top of the ascender of the tallest character in its text line, if any) or use **B** (or **b**) for baseline alignment of the table (the default, where the created table aligns with the baseline of its line as if it were a text character). For baseline offset, you must specify a value between -11" to 11.5", in the form (*text align, offset*).

item runaround

Unanchored boxes only. Either **I** for item runaround or **N** for no runaround (these are case-insensitive, as are most of the single-character keys).

frame width

(*see page 48*) Width of the table's frame (minimum 0 pt (none); maximum 504 pt).

frame color

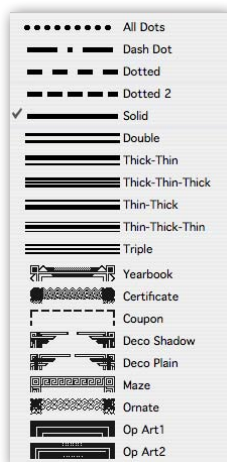
(*see page 49*) Color of the table's frame: either a quoted color name or one of the key characters as described under the **<c>** tag in Chapter 4.

frame shade

(*see page 49*) Shade of the box's frame, as a percentage (minimum 0%; maximum 100%).

frame style

Style of the box's frame, specified as the frame style name in quotes (e.g., "Solid"). Following is a list of the various applications and frame style names.



QuarkXPress frame names



InDesign frame names

InDesign Caveat: Xtags for InDesign accepts QuarkXPress algorithmic names as well as any InDesign frame names. QuarkXPress has several bitmapped styles (**Yearbook** through **Op Art2**) that are mapped to InDesign's **Solid** style, as is the **Dash Dot** style (since there's no InDesign equivalent style). English, French, and German-localized names are supported.

background color

(see page 50) Background color of the table, or **N** for no ("None") color. Note that setting a blend on a table will produce a single blend for the entire table that will only be visible behind transparent cells (cells with **None** background color or those with opacities other than 100%).

background shade

(see page 50) Background shade of the box, as a percentage (minimum 0%; maximum 100%).

text outset

(see page 50) Text runaround exterior margin on all four sides of the table (minimum -288 pt; maximum 288 pt). *Use negative values with caution: it appears to be supported by XPress, but the results and the way it displays aren't reliable.*

gridline width

Width of the table's gridlines (minimum 0 pt (none); maximum 504 pt).

gridline width can take the form of a sublist, allowing you to change some of the gridlines independently: (*top, left, bottom, right, all interior horizontal, all interior vertical*)

gridline style

Style of the table's gridlines, specified as the frame style name in quotes (e.g., "**Solid**"). You can see valid frame styles for both QuarkXPress and InDesign on page 55.

gridline style can take the form of a sublist, allowing you to change some of the gridlines independently: (*top, left, bottom, right, all interior horizontal, all interior vertical*)

gridline color

Color of the table's gridlines: either a quoted color name or one of the key characters as described under the **<c>** tag in Chapter 4.

gridline color can take the form of a sublist, allowing you to change some of the gridlines independently: (*top, left, bottom, right, all interior horizontal, all interior vertical*)

gridline shade

Shade of the table's gridlines, as a percentage (minimum 0%; maximum 100%).

gridline shade can take the form of a sublist, allowing you to change some of the gridlines independently: (*top, left, bottom, right, all interior horizontal, all interior vertical*)

gridline opacity (<v7.00> tags only)

Opacity of the table's gridlines, as a percentage (minimum 0%; maximum 100%; default 100%). This parameter only exists for tags marked as **<v7.00>** (see page 23), and must be omitted otherwise.

gridline opacity can take the form of a sublist, allowing you to change some of the gridlines independently: (*top, left, bottom, right, all interior horizontal, all interior vertical*)

gridgap color

Color of the table's grid gaps (frame backgrounds): either a quoted color name or one of the key characters as described under the **<c>** tag in Chapter 4.

gridgap color can take the form of a sublist, allowing you to change some of the grid gaps independently: (*top, left, bottom, right, all interior horizontal, all interior vertical*)

gridgap shade

Shade of the table's grid gaps, as a percentage (minimum 0%; maximum 100%).

gridgap shade can take the form of a sublist, allowing you to change some of the grid gaps independently: (*top, left, bottom, right, all interior horizontal, all interior vertical*)

gridgap opacity

Opacity of the table's grid gaps, as a percentage (minimum 0%; maximum 100%; default 100%). This parameter only exists for tags marked as **<v7.00>** (see page 23), and must be omitted otherwise.

gridgap opacity can take the form of a sublist, allowing you to change some of the grid gaps independently: (*top, left, bottom, right, all interior horizontal, all interior vertical*)

tab order, link order

Tab order and link order are one of:

- ◆ LRT = (left-to-right top-down)
- ◆ RLT = (right-to-left top-down)
- ◆ TLR = (top-down left-to-right)
- ◆ TRL = (top-down right-to-left)

box name

Optional scripting-relative box name.

layer name

Unanchored tables only. The layer on which the unanchored table will be created and placed. If the layer does not yet exist, it is created with default color and visibility. If the layer cannot be created, the table is placed on the currently selected (active) layer.

Tags for Table Rows

Table row tags (**&trs...&tre**) must be placed between the table start (**&ts** or **&tsu**) and table end (**&tse**) tags. They set both row-specific attributes, and certain cell attribute defaults. Parameters whose names are shown in **bold** have optional, advanced sub-parameters. Parameters generally take their default values from the document's table tool, but those marked with a † have no tool defaults (see the parameter description for defaults assigned by Xtags). Parameters within [brackets] only exist in **<v7.00>** tags—for other input source versions, these parameters should be omitted altogether.

The **table row tag** takes the following form:

```
<&trs(type, height, background color, background shade,
gridline width, gridline style, gridline color, gridline
shade, [gridline opacity,] gridgap color, gridgap shade
[, gridgap opacity])>...cells...&tre8
```

⁸ <&tre> tags are optional. Xtags will automatically infer a row end tag when it encounters another table row start (<&trs>) or a table end (<&tse>) tag.

type

- ◆ **n** (or **N**) = normal row (default)
- ◆ **h** (or **H**) = table header row
- ◆ **f** (or **F**) = table footer row

height

Specifies the row height. **A** will auto-size (which is the default). This parameter can also be expanded to the form: (**A, max height**) to limit the auto-size growth for the row to some maximum.

background color and background shade

(see page 50) These set the defaults for all the table cells contained in this row (minimum 0%; maximum 100%).

gridline width

Width of the row's gridlines (minimum 0 pt (none); maximum 504 pt).

gridline width can take the form of a sublist, allowing you to change some of the gridlines independently: (*top, left, bottom, right, all interior horizontal, all interior vertical*)

gridline style

Style of the row's gridlines, specified as the frame style name in quotes (e.g., "Solid"). You can see valid frame styles for both QuarkXPress and InDesign on page 55.

gridline style can take the form of a sublist, allowing you to change some of the gridlines independently: (*top, left, bottom, right, all interior horizontal, all interior vertical*)

gridline color

Color of the row's gridlines: either a quoted color name or one of the key characters as described under the **<c>** tag in Chapter 4.

gridline color can take the form of a sublist, allowing you to change some of the gridlines independently: (*top, left, bottom, right, all interior horizontal, all interior vertical*)

gridline shade

Shade of the row's gridlines, as a percentage (minimum 0%; maximum 100%).

gridline shade can take the form of a sublist, allowing you to change some of the gridlines independently: (*top, left, bottom, right, all interior horizontal, all interior vertical*)

gridline opacity (<v7.00> tags only)

Opacity of the row's gridlines, as a percentage (minimum 0%; maximum 100%; default 100%). This parameter only exists for tags marked as <v7.00> (see page 23), and must be omitted otherwise.

gridline opacity can take the form of a sublist, allowing you to change some of the gridlines independently: (*top, left, bottom, right, all interior horizontal, all interior vertical*)

gridgap color

Color of the row's grid gaps (frame backgrounds): either a quoted color name or one of the key characters as described under the <c> tag in Chapter 4.

gridgap color can take the form of a sublist, allowing you to change some of the grid gaps independently: (*top, left, bottom, right, all interior horizontal, all interior vertical*)

gridgap shade

Shade of the row's grid gaps, as a percentage (minimum 0%; maximum 100%).

gridgap shade can take the form of a sublist, allowing you to change some of the grid gaps independently: (*top, left, bottom, right, all interior horizontal, all interior vertical*)

gridgap opacity

Opacity of the row's grid gaps, as a percentage (minimum 0%; maximum 100%; default 100%). This parameter only exists for tags marked as <v7.00> (see page 23), and must be omitted otherwise.

gridgap opacity can take the form of a sublist, allowing you to change some of the grid gaps independently: (*top, left, bottom, right, all interior horizontal, all interior vertical*)

Tags for Table Cells

Table cell tags (&tcp and &tcs) must be placed between the table row start (&trs) and table row end (&tre) tags, or "effective" table row end tags (&trs, or &tse). Parameters whose names are shown in **bold** also have more advanced forms which are covered in greater detail in previous subsections. Parameters generally take their default values from the appropriate tool.

The *table picture cell tag* with its basic parameters has this form:

```
<&tcp(width, height, horiz span, vert span, flags,
background color, background shade, placement, picture
scale x, picture scale y, picture offset x, picture offset
y, picture angle, picture skew, picture path name, picture
type)>
```

The *table text cell tag* with its basic parameters has this form:

```
<&tcs(width, height, horizontal span, vertical span,
flags, background color, background shade, text angle,
text skew, text inset, baseline offset, baseline minimum,
vertical alignment, interparagraph maximum)>...text contents...
<&tce>
```

Table picture and text cell parameters, except for those listed below, behave exactly as regular picture and text box parameters (see page 33). The unanchored text cell content environment is just like an unanchored box—you can create anchored and unanchored boxes (and even tables) within it.

As each table cell is entered, the character and paragraph settings will reset to whatever was in effect at the point of the table start (<&ts>) tag.

width, height

width and height parameters are currently ignored, reserved for future use.

horizontal span, vertical span

Number of rows/columns spanned by this cell. Note that a given cell cannot overlap an already-existing spanned cell.

flags

- ◆ **h** (or **H**) = flip horizontal
- ◆ **v** (or **V**) = flip vertical
- ◆ **a** (or **A**) = runaround all sides (text cells only)
- ◆ **p** (or **P**) = suppress picture printing (picture cells only)
- ◆ **w** (or **W**) = allow cell content to wrap (skip cell when auto-sizing the column's width)

All other parameters behave exactly as they do for text and picture boxes. Please see chapter 7 for more details.

Example 8.2

Xtags Code:

```
<&ts(,,2,5,(2"|L''|1.5"|50%\='),r,,2pt,,,,,,2pt)>¶
<&trs(,,,2pt)><&tcs(,,2,,,C,25)><*C><z(. *2)>Famous
Equalities<&tre>¶
<&trs><&tcs>Mathematical constants<&tcs>e<+>i<f"symbol">p<
a$> = -1<&tre>¶
<&trs><&tcs>Definition of Pi<&tcs>p = <V>c<a$>/<V-
>d<&tre>¶
```

9 Table cell end <&tce> tags are optional. Any table tag encountered will end the current cell.

```
<&trs><&tcs>Pythagorean Theorem<&tcs>a<+>2<+> + b<+>2<+> =  
c<+>2<&tre>¶  
<&trs><&tcs>Energy/mass conversion<&tcs>E = mc<+>2<&tre>¶  
<&tse>¶
```

Formatted Result:

Famous Equalities	
Mathematical constants	$e^{i\pi} = -1$
Definition of Pi	$p = c/d$
Pythagorean Theorem	$a^2 + b^2 = c^2$
Energy/mass conversion	$E = mc^2$

Chapter 9

Applying Master Pages

Xtags supports tags to apply a master page to the current document page or spread.

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

The `&m` and `&mf` tags apply the designated page to the current page or spread in the case of a double-sided document. The `&mp` and `&mpf` tags apply the corresponding page (left or right) of the designated master to the current page (and *not* to the entire current spread in the case of a double-sided document).

The concept of “current page” is a little tricky. It’s the page on which you’d find yourself if you were to interactively place the insertion point right after the text imported up to the start of the particular `&m`, `&mp`, `&mf`, or `&mpf` tag under consideration. Note that applying a master page can actually change which page you’re on, depending on the size and number of text boxes linked into the automatic text flow on the old and new master pages, and the amount and kind of text involved. It’s best to only use these tags when you or your Xtags-generating application are fully aware (or in control) of page boundaries (either forced with new box special characters `<\b>` or based on a line count or on general knowledge of where the master page application tag will fall with respect to page breaks).

The `masterpage` parameter is either a number, where master page A is 1, master page B is 2, and so on, or a (quoted) master page/spread name such as `Master A` or `Chapter Start`, as displayed and editable in the text field of the QuarkXPress **Document Layout** palette, or the master page name with its initial lettered prefix: `A-Master A`.

The last `&m` or `&mp` tag found on a page is the one used if more than one `&m` tag appears on that page. Conversely, the first `&mf` or `&mpf` (`f` is for “master [page], first”) appearing on a page (or spread for `&mf`) takes precedence over any other `&mf` tags occurring in the same page/spread. If both an `&m` or `&mp` and an `&mf` or

`&mpf` tag appear on the same page (or spread, as applicable), then the final `&m` or `&mp` tag “wins.”

Example 9.1

Xtags Code:

```
@$:Because it's difficult to show master page usage in a
manual, our example uses boxes, shown here as mini pages.
The default master page is Master A.¶
<\b><&m"Master C">Once we force our way to page 2, the
<B>&m<B> tag will apply master page C to it. Then, we will
once again force a new page break.¶
<\b><&mf"Master A">The third page has three master page
tags on it. The final <B>&m<B> takes precedence over all
the others.<&m"Master C">An <B>&mf<B> tag determines the
master page <BI>only<BI> when there are no <B>&m<B> tags
on the same page.<&m"Master B">¶
```

Formatted Result:

PAGE 1	PAGE 2	PAGE 3
Master A	*Master C*	*Master B*
Because it's difficult to show master page usage in a manual, our example uses boxes, shown here as mini pages. The default master page is Master A.	Once we force our way to page 2, the <code>&m</code> tag will apply master page C to it. Then, we will once again force a new page break.	The third page has three master page tags on it. The final <code>&m</code> takes precedence over all the others. An <code>&mf</code> tag determines the master page <i>only</i> when there are no <code>&m</code> tags on the same page.

Chapter 10

Working with Translation Tables and Macros

Translation Tables

The Xtags translation table facility allows you define a e specifying one or more string substitutions—*translations*—to be performed by Xtags on incoming text. Xtags performs these translations prior to any tag interpretation, so this facility may be used to substitute XPress Tag or Xtags constructs for other products' tags, or for site-specific generic tags. However, this translation facility is perfectly general, and may be used to substitute any string for any other string; e.g., some sites have used it to perform elaborate quote conversion beyond the simple quote conversion capabilities of XPress Tags and Xtags.

Using the Translation Table Tag

`<&tt2"table name">`

This tag tells Xtags to start using the specified translation table. It also makes Xtags forget about any translation table currently in effect, since it may appear at any point in an input file, even multiple times with different table names (although in most cases it would appear once, at the start of input).

`<&tt2w"table name">`

This is identical to the `&tt2` tags, except that it's "weak" and thus doesn't provoke an error if the called-for translation table file isn't found. This can be useful if you'd like to invoke a translation table if it's present, but the results don't necessarily depend on it, if it's not.

The string parameter *table name* must be the name of a file. Xtags for QuarkXPress looks for this file in the following folders and in the order given:

1. the same folder as the current QuarkXPress document;
2. the Xtags sub-folder of the QuarkXPress folder;

3. the folder `~/Library/Preferences/Xtags` under Mac OS X, or in the working directory under Windows; and
4. an Xtags sub-folder of any path specified in the `emxtagspath` environment variable under Windows.

InDesign has a different folder search order:

1. the folders `~/Library/Preferences/Xtags` and `/Library/Preferences/Xtags` under Mac OS X, or `../user/My Documents/Xtags` under Windows; and
2. the `/Network/Library/Preferences/Xtags` folder under Mac OS X, or `../All Users/Documents/Xtags` under Windows; and
3. the same folder as the current QuarkXPress (or InDesign) document;
4. the same folder as the tagged text file being imported (if a tagged text file is being imported).

Translation Table Format

A translation table is an ordinary text file created by any application (be sure to save the text as a raw text file if you create it using a word processing or spreadsheet application). The first line of every translation table file is a header line, and the very first character in this first line is defined to be the comment character for that translation table. If the line is empty, then no comment character is defined, which is often desirable.

For example, if a translation table contained the following first line, then its comment character would be a semicolon:

```
; Translation table for parts catalogue.¶
```

Any text following the comment character within the lines of the translation table is ignored by Xtags, up through the end of its line. *Note that this renders the comment character unusable in translations, so choose it carefully (or leave it undefined by leaving an empty first line).*

If the second character of the first line in a translation table is `e`, then the the second and third characters are interpreted as specifying the desired character-set, enabling a given translation table to be used either under Mac OS X or under Windows with automatic character set translation. Unicode is also supported. The character set codes are the same as for the XPress Tags `<en>` character set selection tag (see reference list on page 22).

Translation specifications

The remainder of a translation table file should contain lines (separated by carriage returns, with trailing line feeds ignored) of the following form:

```
source→target→comments¶
```

with the three elements separated by one or more tabs (but no leading or trailing blanks, which are significant).

Source is a string which may appear in the Xtags input file, and *target* is another string into which Xtags should change each occurrence of *source* that it finds.

Target is often an Xtags sequence, but can be anything, including nothing (in which case the source string will be silently elided from all input)—“nothing,” in this case, being specified as a line with just a *source* string on it.

Comments is entirely optional, and is completely ignored. Empty lines are ignored (and, in particular, lines beginning with the comment character are considered empty). Literal *source* and *target* strings are limited to 4096 characters in length, but there is no limit (other than available memory) on the number of translations in a given table.

After a **&tt2** tag, source strings are found and replaced, ignoring any input file line boundaries—i.e., the translation facility is purely character-stream-oriented, not line-oriented. Source strings are case sensitive, and in the case of a string in the imported file matching more than one source string in the translation table, the translation with the longest matching string is chosen. A translation, once made, is not re-scanned to see if it matches any other translation's source string—Xtags just substitutes the original target string for the matched source string and goes back to processing input normally (including looking for source strings), starting with the input character following the matched source string.

The following special sequences in source strings are supported to specify characters that otherwise couldn't be entered, given the constraints of translation table syntax:

<code>\b</code>	backspace (ascii 8)
<code>\t</code>	tab (ascii 9)
<code>\n</code>	newline/linefeed (ascii 10)
<code>\r</code>	carriage-return/return (ascii 13)
<code>\v</code>	vertical tab
<code>\\</code>	literal backslash character (an actual backslash in a translation source string must be “escaped” by doubling it)

The special forms `<\#9>` and `<\#13>` (as well as any other XPress Tags) may be used in target strings to place tabs and paragraph end characters within a target translation.

Generally, source strings should be enclosed in some sort of brackets (angle brackets are not recommended, however, because of the potential for confusion, should a given translation be forgotten), or at least start with some little-used character.

Some typical bracketed source strings might look like:

<code>«MDNM»</code>	chevron marks
<code>{BP}</code>	curly braces
<code>[ep]</code>	square brackets
<code>%SIMS%</code>	percent signs

Although Xtags places no restrictions on what characters may appear in source strings, for efficiency's sake, the first character of each source string should usually be a character that does not otherwise appear very often (or at all) in the imported file.

Because translations operate at the very lowest level (before Xtags ever does any tag interpretation), avoid using translations that might confuse Xtags' tag interpretation. For example, if you define a translation as:

```
\\b      •
```

in a reasonable attempt to create a shorthand for a bullet, then any use of the special new box character `<\b>` will cause an error, as Xtags will only see a `<•>` after the translation occurs. It's better to avoid the following characters in a source string, unless they're used in conjunction with enough other characters to guarantee that they won't clash with ordinary Xtags tags:

```
< > @ # & \ : = . ; , / * + -
```

Here's an example of a translation table in use.

Example 10.1

Translation Table “Example TT”:

```
; Remember that the first character of the first line defines¶
; your comment character; all lines in this table starting¶
; with a semicolon are comments and ignored by Xtags.¶
; look for          replace      explanation¶
!b                  •            bullet¶
!B                  ß            beta¶
```

```

!d          †          dagger¶
!dd         ‡          double dagger¶
; Only allow explicit [ep] tags to denote end of paragraph,¶
; not returns.¶
\r          <\#32>     return becomes a space¶
[ep]\r      <\#13>     ep tag at end of line becomes¶
;           paragraph end¶
\[ep]       [ep]      \[ep] becomes literal version¶
;¶
; Define some tag bracketing pairs for "logical"¶
; tagging.¶
[a]         <B>        author begin¶
[/a]        <B>        author end¶
[t]         <I>        title begin¶
[/t]        <I>        title end¶
; Following is a source string that turns into nothing.¶
[empty]     gets deleted¶

```

Xtags Code:

```

<&tt2"Example TT">@$:First note how the¶
carriage returns in the input¶
are turned into spaces, and only the explicit <B>\[ep]¶
<B>tag becomes an end of paragraph.[ep]¶
Also note how the literal end paragraph¶
tag target is not re-scanned, once¶
it is substituted, or else we'd have an end¶
of paragraph after "the explicit" above.[ep]¶
Now, let's use a bullet !b, a beta !B,¶
a dagger !d and a double dagger !dd.¶
These examples show the case sensitivity of¶
source strings (the bullet and beta¶
differing only in case), and the longest-match¶
rule (the double dagger source¶
string is matched, instead of just the dagger¶
source string, followed by a "d").[ep]¶
Notice how a string "[empty]" can¶
turn into nothing.[ep]¶
Here's a book reference, [t]Gilbert Keith¶
Chesterton[/t] by [a]Maisie Ward[/a], tagged¶
with logical tags.[ep]¶

```

Formatted Result:

First note how the carriage returns in the input are turned into spaces, and only the explicit [ep] tag becomes an end of paragraph.

Also note how the literal end paragraph tag target is not re-scanned, once it is substituted, or else we'd have an end of paragraph after "the explicit" above.

Now, let's use a bullet •, a beta ß, a dagger † and a double dagger ‡. These examples show the case sensitivity of source strings (the bullet and beta differing only in case), and the longest-match rule (the double dagger source string is matched, instead of just the dagger source string, followed by a "d").

Notice how a string "" can turn into nothing.

Here's a book reference, *Gilbert Keith Chesterton* by **Maisie Ward**, tagged with logical tags.

Adding Entries to a Translation Table

Xtags supports the tag `&tte("source", "target")` to allow you to add new translation table entries directly in your Xtags input file, taking immediate effect. If there is already a translation table in effect (instantiated by the `&tt` or the `&tt2` tag), the new source-to-target translation is simply added to the table. If there is no translation table in effect, one is created, and the new translation is entered as the first in the table.

Source is the source translation string, and *target* is the target translation string just as in the `&tt2` translation file invocation tag. You can include as many target strings as you like, and they'll be considered one long target (to get around the Xtags list parameter string length limit of 255 characters). For example, `&tt2e("%param1%", "This ", "is ", "a ", "target string")` will define a target string **This is a target string** that will replace the source string `%param1%` in subsequent input. The source string is limited to 255 characters in length, but the built-up target string can be as long as 16,383 characters in length.

The intent of this tag—though you can use it for whatever you desire—is to allow you to easily parameterize subsequent Xtags code (which can thus be "boilerplate" with parameters of your own devising, though the more obscure the parameter name, the less likely that the boilerplate will be disturbed by Xtags' blind source-to-target string translations).

Unfortunately, you can't re-define the translation for a given source string (the second and subsequent definitions are entered but ineffectual).

In the `&tt2e()` variant of this tag, you can use the same escape characters in the source string as are supported in the translation file invocation tag, namely, `\r` (for ASCII carriage return), `\n` (new line), `\t` (tab), `\b` (backspace), `\\` (a literal backslash) (anything else after a backslash is ignored). For example, to trans-

late a return to the literal `{return}`, you'd use something like `&tt2e("\\r", "{return}")`.

As an example, consider the input:

```
<&tt2e("%1%", "first")>This is the %1% attempt.¶
```

Importing it will result in the following output:

This is the first attempt.

Turning Off Tag Interpretation

When debugging a translation table, it is often useful (or vital) to see the intermediate tags being generated by Xtags before they are interpreted and transformed into formatted text. The `&d(0)` debugging tag turns off all tag interpretation. For example, the tags

```
<&tt2"...& d(0)> ...more text and tags...¶
```

would result in the rest of the text and tags being translated with no further tag interpretation by Xtags (as if you had unchecked **Include style sheets** in the original **Get Text with Xtags...** dialog). Note that there is no tag to turn tag interpretation back on once you've turned it off, so this is really only useful when debugging.

Macros

Macros are like subroutines in a programming language—they save typing for complex (usually parameterized) tag sequences. They encapsulate complex thoughts in a simpler interface such that you can change the macro source but not change your tag sources, to accommodate change. For example, if you have a fraction macro, you can mark up your tagged text with the macro calls, confident that even if you decide to change the way you build fractions, your macro-using tagged source text won't have to change.

Macro Definition and Invocation Tags

The `&!` tag is used to define a macro, like this:

```
<&!(macroname, macrobody)>
```

This tag defines a macro named *macroname* which executes the substitutions in its body *macrobody*.

After a macro is defined, whenever the macro invocation tag “!” is encountered, the *macrobody* is substituted for the whole macro invocation tag, with any given arguments (there may be none or as many as you like) substituted for argument references in the original *macrobody*:

```
<!macroname(arg1, arg2, arg3, ..., argn)>
```

Note that spaces before and after the arguments are ignored, and any multiple spaces in the arguments are compressed to one, unless you enclose exactly what you want in double quotation marks (matching curly or straight).

Macrobody as used above may consist of any number of parameters, and Xtags will append the second through the last into one long body, up to 4,096 characters' worth. Thus, the macro definition `<&!(m1,">fee", "fie", "fo", "fum<")>` is the same as `<&!(m1,">feefie", "fofum<")>`, which is, in turn, the same as `<&!(m1,">feefiefofum<")>`. (This feature exists to work around the 255-character limit of the individual list parameters in the macro definition tag.)

Argument references in *macrobody* are of the form `!n`, where *n* is a positive number, with `1` referring to the first argument of the macro, `2` referring to the second, and so on. Argument references may also use only a portion of the argument, using the following forms:

- `!i` the entire *i*th argument, for *i* from 1 through 9 (single digit)
- `!(i)` the entire *i*th argument, for any value of *i*
- `!(i j)` the *j*th character of the *i*th argument
- `!(i j:)` the *j*th through the last character of the *i*th argument
- `!(i j:k)` the *j*th through the *k*th character of the *i*th argument

One or more spaces must separate the argument number from any first character number, and a colon must separate the first from the last character number (any extraneous spaces are ignored). If *j* or *k* are zero or negative, then they are character indices counting back from the end of the argument *n* (so `0` means “the last character of the argument”).

Any exclamation marks (!) in *macrobody* must be doubled (!!) to avoid being interpreted as argument reference prefixes.

No translations are performed on the *macrobody* in a macro invocation (though the definition and invocation themselves may be the result of a translation). A *macrobody* can't contain other macro invocations; nested macro invocations aren't supported.

Note that the *macrobody* starts off being interpreted as Xtags commands, and certainly should end up that way, so if you need to actually create some textual input, the *macrobody* will have to contain a `>` and then a subsequent `<` tag, to leave tag mode and later re-enter it. In this case, you'll have to surround the *macrobody* with double quotes (straight or curly) to keep the `>` and `<` from confusing Xtags.

Xtags supports the use of quote characters in macro bodies (which are just strings), so you can define macros that build boxes and otherwise use tags that require quoted strings. For example, the macro

```
<&! (build, "&pbu2(0,0,2\",1\",,,,,,,,,,1,,,1,c,,,,,,,,\"!1\",
,)&tbu2((0,b1,1),0,2\",(1\",s))>@caption:!!<&te)>
```

when invoked with `<!build("test.tif")>`, will expand to:

```
<&pbu2(0,0,2",1",,,,,,,,,,1,,,1,c,,,,,,,,,"test.tif",,)
&tbu2((0,b1,1),0,2\",(1\",s))>@caption:test.tif<&te>
```

(We've wrapped the output for readability.) This tag sequence will build an unanchored picture box, 2" by 1", at the upper left corner of the current page, filling it with the picture named by its argument (test.tif), and then place a text box at its lower left corner of the same size (using relative box placement syntax), filling the text box with the name of the picture file in a caption style, and shrinking the height of the text box to fit the picture name (again, using a height parameter sub-list specification for shrink-to-fit).

Separator-specifying macro definition tag

Xtags has a separator-specifying macro definition tag `<&!2(macroname, argument separator, body1, body2, ..., bodyn)>`, which parallels the `&!` tag but whose second argument is a string whose first character is used as the argument separator character when the given macro is invoked.

For example, the tag sequence

```
<&!2(frac, "/", ">!1 divided by !2<")><!frac(3/4)>
```

would result in the output **3 divided by 4**. (This example should also give you some idea of why we added this tag.)

Example 10.2

The following macro definitions and invocations produce the indicated results. Note the initial and final `>` and `<` are to leave tag command mode and subsequently reenter it.

```
<&! (m1, ">Macro 1 here!!<")!m1())>¶
```

which becomes, after macro processing:

```
< >Macro 1 here!< >¶
```

which further becomes, after tag processing:

```
Macro 1 here!
```

The following macro definition and invocation display the use of simple argument references (of both forms):

```
<&! (macro, ">Arg 1 is «!1», Arg 2 is «!(2)»<") :
!macro(one, two)>¶
```

which becomes:

```
Arg 1 is «one», Arg 2 is «two»¶
```

after all the `<:>` and `<>` tags are processed.

More complex argument references, extracting parts of a macro argument, are illustrated by the following macro (which we've wrapped for clarity):

```
<&! (complex, ">Arg 1 is «!1»,
char 1 is «!(1 1)»,
last char is «!(1 0)»,
chars 2 to 5 are: «!(1 2:5)»,
chars 3 to last are: «!(1 3:)» and: «!(1 3:0)»,
next-to-last two chars are: «!(1 -2:-1)».<")
!complex("1234567890")>¶
```

which becomes

```
Arg 1 is «1234567890»,
char 1 is «1»,
last char is «0»,
chars 2 to 5 are: «2345»,
chars 3 to last are: «34567890» and: «34567890»,
next-to-last two chars are: «89».¶
```

As a real-world example, here's a fraction macro which you can embellish to generate your favorite style of fraction:

```
<&! (fm1, "+>!(1 1:-1)<k-10>!(1 0)<+k-5>/!(2 1)<k-10>!(2
2:)<")>¶
@$:<! fm1(1,2) <! fm1(3,200) > 4<! fm1(3,8)>
<! fm1(32,457)>¶
```

The argument references read, respectively, the first through the next-to-last character of argument 1, the last character of argument 1, the first character of argument 2, and

the second through the last character of argument 2. It produces the following Xtags code:

```
<+><k-10>1<+><z9><k-5>/2<z11><k-10>    <+>3<+><z9><k-5>/2<k-10>00<z11>    4<+>3<+><z9><k-5>/8<z11><k-10>
<+><k0>3<k-10>2<+><z9><k-5>/4<k0>5<k-10>7¶
```

which gives us a fairly reasonable-looking fraction, kerning the final character of the first argument (2) against the virgule, and kerning the virgule against the first character of the second argument (4). Here is the full output, as it actually appears:

$\frac{1}{2}$ $\frac{3}{200}$ $4^3/8$ $\frac{32}{457}$

This fraction macro assumes that the superscript VScale and HScale settings in QuarkXPress's preference dialog are set at 80% and the Offset is set at 33%. You could accomplish the same effect with explicit z and b tags.

Note that if you're using a translation table, you don't need to put all of your macro definitions in every input file. A better method would be to have a translation entry for the string {prolog} that mapped to a series of macro definitions. Then, you could include text like this in your input file:

```
<&tt2"table">{prolog}¶
```

Even better, if you define some unlikely string as the starting and ending brackets for a fraction invocation, e.g., {{ and }} with the translation table lines:

```
{prolog} <&! (fm1, "...")>           define macros
{{      <! fm2 (                       fraction macro begin
}}      )>                             fraction macro end
!b      <f"Zapf Dingbats"><z14>o<f$z$><\#9>  bullet / tab
```

then your input file can specify fractions in a more friendly fashion:

```
<&tt2"table">{prolog}
@step: !bNext, add {{1,2}} cup of flour to the mixture and then slowly
pour in {{3,4}} cup of scalded milk.¶
```

which yields:

- Next, add $\frac{1}{2}$ cup of flour to the mixture and then slowly pour in $\frac{3}{4}$ cup of scalded milk.

Chapter 11

Automating Document Building

Xtags Simple Batch Facility

Xtags has a batch-processing facility for automated (“hands-off”) document building and extracting. Xtags can watch for control files to appear in a watched folder, and then obey those control files to import Xtags files into either the current document or a selected template, or obey those control files to export from the current document.

When Xtags starts up, it looks for an input subfolder in the QuarkXPress folder called Xtags/batch (Mac OS) or Xtags\batch (Windows OS) .

If this folder is found at start up, Xtags checks once per second during the current XPress session (when XPress is otherwise idle) for batch control files—files of any name ending in .xbc. Xtags will then process each such file found as follows. If this folder is not found at start-up, batch processing is disabled for the current XPress session.

If the batch input file can't be opened because it's “busy” (opened for writing somewhere else), Xtags simply backs off and gives up on it (on the assumption that it's being created, and is not available for input yet). This is the basis of a simple interlock mechanism: you (or your external program) should create or otherwise put in place any files mentioned in the batch control file, then create the batch control file with write access (which is normal), and then close it. (Or, you could create the file without the .xbc extension and rename it to have the .xbc extension when it's ready to use.)

To process a given batch control file once it's been successfully opened, Xtags reads five text lines from the file, each with the following respective meaning (where “none is given” means the respective line is empty):

document template path

A full path to a document template to open. Xtags will open the given document and attempt to find and select the auto-flow text box on the first page. If none is given, then any current document is used, and, again, Xtags attempts

to find and select the auto-flow text box on the first page. If there's no current document, and no *document template path* is given, then the Xtags batch process will silently fail.

Also, if no auto text flow box can be found, the Xtags batch process will silently fail—unless there is no input file and the **A** “output all stories” flag is used—see *flags* below.

Xtags input file path

A full path to an Xtags input file to read and flow into the current auto-flow text chain. If no input file path is given, nothing is read.

Xtags output file path

A full path to an Xtags output file into which Xtags will write either the current auto-flow text chain contents (no **A** flag given) or all stories in the document (**A** flag given), each prefixed by a comment of the form ‘-*-’ . If no output file path is given, nothing is written.

document output path

A full path to where to Xtags should save the document resulting from the above steps. If no document output path is given, no document is saved.

flags

A list of single-character, case-insensitive flags controlling the above operation. Currently there are only two defined flags: **C**, which means “close the resulting document at the end of batch processing (whether errors occurred or not)”, and **A**, which means “process all stories for output.”

Once the batch input file processing is finished (successfully or not), the batch input file is deleted.

Using Scripts to Automate Document Building

Xtags adds tag import/export functionality to all scripting languages supported by QuarkXPress (AppleScript) and InDesign (AppleScript, ExtendScript, and VB Script). Available options are also limited by the OS. (AppleScript is only available on Mac OS computer systems and VB Script can only be run on Windows.) This is a limitation of the language rather than Xtags.

This section assumes some very basic familiarity with scripting, such as why it's useful and how it works at some very high level. At the end of this chapter we will show working examples, written in all three major scripting languages, that can be used to kick-start your own custom scripts.

Get Text with Xtags

AppleScript (Mac-only)

```
get text with Xtags
  from source
  [with/without quote conversion]
  [with/without error reporting]
  [with/without document updating]
  [encoded as characterSet]
```

ExtendScript (InDesign only)

```
<object>.getTextWithXtags (source,
  quoteConversion,
  errorReporting,
  documentUpdating,
  characterSet);
```

VB Script (InDesign/Windows only)

```
<object>.getTextWithXtags source,
  quoteConversion,
  errorReporting,
  documentUpdating,
  characterSet
```

Get text with Xtags interprets the contents of the given Xtags *source* (file alias or string containing tags), replacing the current insertion point or selection with the resulting text. You must first have a document open and a text box selected before invoking Xtags in this fashion from a script.

The *source* parameter may be either a reference to an existing file or a string which itself contains the tagged text. Xtags will always attempt to coerce *source* to a file reference first, even if *source* is a text type. If you want *source* treated as a string, be sure the text in the string does not resolve to a valid and existing file reference.

The boolean parameters *quoteConversion*, *errorReporting*, and *documentUpdating* correspond to the options in the **Get Text with Xtags...** dialog. You must pass either **true** or **false** for each of these in ExtendScript or VB Script. In AppleScript, if you don't specify a particular boolean parameter, Xtags will use the corresponding preference item. Thus, if you want to be sure to control Xtags' behavior exactly, specify *all* boolean parameters.

characterSet gives the character coding as an enumerated value. Valid options are given below.

Save Text with Xtags

AppleScript (Mac-only)

```
save [text] with Xtags
  [to alias]
  [encoded as characterSet]
  [with/without style definitions]
  [with/without separate tags]
  [with/without full list elements]
```

[Result: file name]

AppleScript (Mac-only)

```
<object>.save[Text]WithXtags (destination,
  characterSet,
  styleDefinitions,
  separateTags,
  fullListElements);
```

VB Script (InDesign/Windows only)

```
<object>.save[Text]WithXtags destination,
  characterSet,
  styleDefinitions,
  separateTags,
  fullListElements
```

Save text with Xtags and **save with Xtags** are different commands:

save text with Xtags may be used with either application or document objects to produce the Xtags text representation for the current text selection (or the whole containing current text box, if requested) or the current picture, line or group box, either saving the results in a given file or simply returning it as a string.

save with Xtags works similarly, but can be used on any exportable object: master spreads, layers, spreads, pages, stories, page items, and any "text" object (like a selection, paragraph, column, etc.). This command ignores the selection and exports the whole referenced object.

Scripts can use either tagged text strings or files:

In AppleScript, without a *to filespec* parameter, the AppleEvent returns the current selection turned into tags as a string. Note that if you supply a file specification with the *to alias* parameter, you shouldn't use the *alias (file-*

spec) form unless the file already exists (it will be overwritten); instead, just supply a string containing the file specification, and Xtags will create the file.

In ExtendScript and VB Script, passing an empty string as the *destination* will cause the tags to be returned as a string.

`CharacterSet` is an enumeration that allows you to set the encoding for imports and exports. See the `<e>` tag for more specifics about how this parameter is used. The enumeration takes a different form in each of the supported languages:

	ExtendScript	VB Script	AppleScript
Win Latin	<code>XtagsTextEncoding.winlatin</code>	<code>idwinlatin</code>	<code>winlatin</code>
Mac Roman	<code>XtagsTextEncoding.macroman</code>	<code>idmacroman</code>	<code>macroman</code>
UTF-8	<code>XtagsTextEncoding.utf8</code>	<code>idutf8</code>	<code>utf8</code>
UTF-16	<code>XtagsTextEncoding.utf16</code>	<code>idutf16</code>	<code>utf16</code>

The remaining three boolean parameters specify whether style definition tags are included or not, whether every tag is output separately or not and whether default parameters to tags are included or not, mirroring the corresponding check boxes in the **Xtags Preferences** dialog.

Example 11.1

Here is a simple example use of this AppleEvent to import text from a file (we've allowed the lines of the script to wrap for clarity, showing explicit line endings). This example imports the file `Adcopy` in the folder `Text` on the disk `Themis` into the current document at the current insertion point, replacing any selected text:

AppleScript (Mac-only)

```
tell application "QuarkXPress"
    activate
    get text with Xtags from alias "Themis:Text:Adcopy"
    without error reporting with document updating
end tell
```

ExtendScript (InDesign only)

```
app.getTextWithXtags("Themis:Text:Adcopy", true, false,
    true, XtagsTextEncoding.utf8);
```

VB Script (InDesign/Windows only)

```
app.ActiveDocument.getTextWithXtags "Themis:Text:Adcopy",
    true, false, true, idutf8
```

Here is a more complicated script which we've annotated (we've again allowed the lines to wrap for clarity).

AppleScript (Mac-only)¹⁰

```
tell application "QuarkXPress"
    activate
    --Prompt for the QuarkXPress template file.
    set templatefile to (choose file with prompt "Select
        template:" of type {"XTMP", "XDOC"})
    --Prompt for the Xtags file to be imported.
    set xtagsfile to (choose file with prompt "Select
        Xtags input file:" of type {"TEXT"})
    open templatefile use doc prefs yes
    --The script assumes that the first box on the page is
        the automatic text flow (we could also use a pre-
        named text box).
    --Set that as the target location.
    set selected of text box 1 of page 1 of document 1 to
        true
    --Attempt the import, catching any resulting error.
    try
        --Insert the file with Xtags, using sensible settings
        get text with Xtags from xtagsfile with quote
            conversion and error reporting without document
            updating
        --If there was an error, display the error message
        on error msg number num
            display dialog "Get text with Xtags failed: "&
                msg & "[" & num & "]"
        end try
        --Additional actions, such as saving the document,
            could go here.
    end tell
```

ExtendScript (InDesign-only)

```
// Prompt for the InDesign template file.
var templatefile = File.openDialog ("Select template:",
    null, false);
// Prompt for the Xtags file to be imported.
var xtagsfile = File.openDialog ("Select Xtags input
    file:", null, false);
var doc = app.open(templatefile);
// The script assumes that the first box on the page is
```

¹⁰ This example was written for QuarkXPress. Running it in InDesign requires some minor modifications.

```

// the automatic text flow. (we could also use a ¶
// pre-named text box). Set that as the target location.¶
app.select(doc.textFrames[0].insertionPoints[0],
           SelectionOptions.replaceWith);¶
// Attempt the import, catching any resulting error.¶
try {¶
    // Insert the file with Xtags using sensible settings¶
    doc.getTextWithXtags(xtagsfile, true, true, false,
                        XtagsTextEncoding.winlatin);¶
}¶
catch(msg) {¶
    // If there was an error, display the error message¶
    alert("Get text with Xtags failed: " + msg);¶
}¶
// Additional actions, such as saving the document, could¶
// go here.¶

```

VB Script (InDesign/Windows only)

```

set fileDlg = CreateObject("UserAccounts.CommonDialog")¶
' Prompt for the InDesign template file.¶
set appcs3 = CreateObject("InDesign.Application.CS3")¶
if fileDlg.ShowOpen = 0 then MsgBox("No file selected:
                               quitting.")¶

templatefile = fileDlg.FileName¶
' Prompt for the Xtags file to be imported.¶
if fileDlg.ShowOpen = 0 then MsgBox("No file selected:
                               quitting.")¶

xtagsfile = fileDlg.FileName¶
set doc = appcs3.open(templatefile)¶
' The script assumes that the first box on the page is¶
' the automatic text flow. (we could also use a pre-named¶
' text box). Set that as the target location.¶
appcs3.select (doc.textFrames(1).insertionPoints(1))¶
' Attempt the import. Insert the file with Xtags, using¶
' sensible settings¶
appcs3.ActiveDocument.getTextWithXtags xtagsfile, true,
                                       true, false, idutf8¶
' Additional actions, such as saving the document, could¶
' go here.¶

```

Chapter 12

Creating Xcatalog and InCatalog Links

First, to make use of or even any sense of these tags, you must be using Xcatalog (for QuarkXPress) or InCatalog (for InDesign). The same link creation tags are used for both products.

Xtags supports three link creation tags:

<&Cs>

Delimits the start of a text selection link.

<&Ce(*dd, flags, key, key type, field, subfield, picture position, price style*)>

Delimits the end of a text selection link and supplies the link information.

<&Cb(*relative box reference, dd, flags, key, key type, field, subfield, picture position, price style*)>

Creates a box-based link for a relatively-referenced box.

The **&Cb** tag creates a box-based link for the box referenced by *relative box reference* (**1** for the most-recently-created box, **2** for the second-most-recently-created box, and so on); this parameter must be within the range of **1** to **99**.

The remaining parameters to **&Ce** and **&Cb** are as follows:

dd

Required name of the data descriptor (what you would have in force, had you been creating this link manually with the linking palette).

flags

One or more of **P** for price, **T** for tagged text, and **U** for unquoted tagged text (**U** implies **T**) (no default).

key

Key value for the link (no default).

key type

Key type code: **L** for “Key from link”, **G** for “Key from group”, **B** for “Key from text (backwards)”, **F** for “Key from text (forwards)”, and **C** for “Key from contents” (default is **L**).

field

Name of the linked field (no default).

subfield

0 or empty—denoting no subfield for the link—or the **1**-based index of the FileMaker subfield for the link.

picture position

Picture positioning code: **S** for “As-Is”, **M** (the default) for “Upper Left” (manual placement, to parallel the **&pb** and **&pbu2** tags default picture placement), **C** for “Centered”, **F** for “Fit to box”, **A** for “Fit to box maintaining aspect ratio” (determines the scale in each dimension that would just fit the box and uses the smaller of the two, then centers the picture in the box), and **L** for “Fill box maintaining aspect ratio” (determines the scale in each dimension that would just fit the box, uses the larger of the two, then centers the picture in the box). These positions corresponding to the six choices in the linking palette pop-up menu for picture placement.

price style

Name of the price style for the link (no default).

dd, *key*, and *field* must be non-empty, and all three must also be shorter than 64 characters in length.

For example, if you wanted to tag a price with a link to your “Catalog” DD, key “1-101”, field “Sale price”, you’d use a tag sequence like:

```
<&Cs>$1.99<&Ce("Catalog", p, "1-101", , "Sale price", , , )>
```

or

```
<&Cs>$1.99<&Ce(Catalog, p, 1-101, , Sale price)>
```

The second form omits unneeded quotes and trailing parameters.

Similarly, if you wanted to tag a newly-created picture box with a link to your “Catalog” DD, with a key from group (placeholder “*”—it doesn’t really matter what you use as long as it’s not empty, though you should avoid the use of curly braces **{}**), as they have a special meaning to Xcatalog/InCatalog), field “picture path”, with “as is” positioning, you’d use a tag sequence like:

```
<&pbu2(...)&Cb(1, Catalog, , *, g, "picture path", , s)>
```

Adding Xcatalog/InCatalog Links to Anchored Boxes

Note that you can add an Xcatalog link to the most-recently-created anchored box (text or picture) with the **&Cb(0, ...)** tag.

We recommend putting the link tag immediately after the anchored box tag. E.g.:

```
... <&tb(...)>...text box contents...<&te><&Cb(0, ...)>  
..¶
```

Chapter 13

Application-specific Tags

Xtags has been designed to allow easy transition between application platforms; however, QuarkXPress and InDesign each contain features that are not available in the other. Xtags provides application-specific tags for these features. If you use these tags, your tagged text will require modifications before being used in the other product.

QuarkXPress-specific Tags

Xtags for QuarkXPress supports the import and export of all XPress tags. Tags not explicitly mentioned in this manual are not supported under InDesign—this includes the QuarkXPress index tags (**xo**, **xc**, and **xi**). (When using these tags, QuarkXPress's Index XTension should be present and the Index palette open, otherwise all index tags will be lost on input or output with Xtags, the same as with Quark's XPress Tags filter.) If Xtags encounters a QuarkXPress index tag while importing into InDesign, it will generate an error.

Xtags for QuarkXPress also supports the hidden text tag (**<A>**) and the paragraph hyphenation and justification tag (**<*h>**). These will be silently ignored if encountered during an import into an InDesign document.

InDesign-specific Tags

Xtags for InDesign supports the InDesign Tags language directly by allowing native InDesign Tagged text to be passed through to the application. This is done by placing it inside another tag: **<&it"...">**, where the ellipsis is your InDesign Tags tag sequence delimited at its start and end by triple quotes. Any InDesign tag is able to be “escaped” in this way, including the table, hyperlink, and indexing tags.

Here are three points to keep in mind when using the InDesign Tags extension:

1. “Escaping” tags creates overhead, and should be avoided if there is an equivalent native Xtags language element.

2. Xtags automatically prefixes your InDesign tags with the proper start file tag (either **<UNICODE-MAC>** or **<UNICODE-WIN>**).
3. You cannot mix Xtags tags within incomplete InDesign Tags tags.

Example 13.1

Xtags Code:

```
We can import native <&it"..."InDesign tagged text by
enclosing it within the &it tag. (<CTypeface:Italic>This
text, for instance, will be italicized.<CTypeface:>)"
"
<tStart:2,1:0:0<tdct:Text>><coStart:<tcaw:340>><rStart:<
trah:17>><clStart:1,1><pstyle:NormalParagraphStyle>Table
s can even be created by using the InDesign table tag.<cl
End:><rEnd:><rStart:<trah:31.4>><clStart:1,1><pstyle:Norm
alParagraphStyle>Note, however, that you cannot use Xtags
constructs within the table; the table end tag must be
placed within the same &it block.<clEnd:><rEnd:><tEnd:>"
"">"
```

Formatted Result:

We can import native InDesign tagged text by enclosing it within the &it tag. (This text, for instance, will be italicized.)

Tables can even be created by using the InDesign table tag.
Note, however, that you cannot use Xtags constructs within the table; the table end tag must be placed within the same &it block.

Appendix A

Xtags Summary

Character Set and Xtags Version Tags ([see page 22](#))

<e0>	Mac OS character set
<e1>	Windows DTP character set
<e2>	ISO Latin-1
<e8>	UTF-16
<e9>	UTF-8
<vn.m>	Set the tags' version (Xtags will interpret tags differently based on this value)

Character Formatting Tags

<bshift>	Set absolute baseline shift, in points (see page 21)
<brshift>	Set relative baseline shift, in percentage of size (see page 21)
<c"color name">	Set color, by "color name" or C M Y K W (see page 19)
<sshade>	Set shade, in percentage of black (see page 19)
<popacity>	Set opacity, as a percentage (see page 19)
<f"font name">	Set font, by "font name" (see page 19)
<hscale>	Set horizontal scaling, in percentage of size (see page 20)
<kkern>	Set kerning for following two characters, in 1/200 ems. Other valid options (InDesign only) are "Optical" and "Metrics" (see page 20)
<ttrack>	Set tracking, in 1/200 ems (see page 20)
<zsize>	Set size, in points (see page 19)
<yscale>	Set vertical scaling, in percentage of size (see page 20)
<a\$>	Reset all character attributes to the character attributes specified by the currently-applied paragraph style sheet.

<a\$\$> Reset all character attributes to the character attributes specified by the currently-applied character style sheet.

Character Face Tags ([see page 18](#))

<P>	Sets plain face	</>	Toggle strike-through
	Toggle bold	<K>	Toggle all capitals
<I>	Toggle italic	<H>	Toggle small capitals
<O>	Toggle outline	<+>	Toggle superscript
<S>	Toggle shadow	<->	Toggle subscript
<U>	Toggle underline	<V>	Toggle superior
<W>	Toggle word underline		
<\$>	Set to current paragraph style sheet's face		
<\$\$>	Set to current character style sheet's face		

Character Ligatures Tag ([QuarkXPress 7.x only. See page 22](#))

<G0>	Disable ligatures
<G1>	Enable ligatures
<G\$>	Set ligatures to paragraph style
<G\$\$>	Set ligatures to character style

Special Character Tags ([see page 22](#))

Tags marked with a † may be made non-breaking if the backslash is followed by an exclamation point (!). For example, the tag <\!s> places a non-breaking space into the document.

<\n>	Insert line break ("soft return", not paragraph return)
<\d>	Insert discretionary (optional) line break
<\->	Insert hyphen
<\m>†	Insert em space (<v7.00> tag mode only); Insert m dash (tags prior to <v7.00>)
<\i>	Insert "indent here" marker
<\t>	Insert right indent tab (not a regular tab—see below)
<\s>†	Insert standard space
<\f>†	Insert flex space (<v7.00> tag mode only); Insert figure (en, half-em) space (tags prior to <v7.00>)
<\p>†	Insert punctuation space
<\q>†	Insert quarter-em (flexible) space
<\h>†	Insert discretionary (optional) hyphen

<code><\2></code>	Insert previous text box page number
<code><\3></code>	Insert current text box page number
<code><\4></code>	Insert next text box page number
<code><\c></code>	Insert new column (force column break)
<code><\b></code>	Insert new box (force box break)
<code><\@></code>	Insert at sign (@)
<code><\<></code>	Insert left angle bracket (<)
<code><\>></code>	Insert right angle bracket (>)
<code><\ ></code>	Insert backslash (\)
<code><\e></code>	Insert “End Nested Style” character. (InDesign only)
<code><\#nnn> †</code>	Insert special character, where <i>nnn</i> is a decimal value representing a character from either MacRoman (if the encoding is set to <e0>) or WinLatin (if the encoding is set to <e1>). If the encoding is neither <e0> nor <e1> (see <e> tag on page 22), then the character set will be selected from the platform default. The most common uses for this tag are <\#13> (paragraph return) and <\#9> (tab)

<code><\#Unnnn></code>	
<code><\#U+nnnn></code>	Insert Unicode special “character”, where <i>nnnn</i> is a 4-digit hexadecimal character code point like <\#U2122> (the ™ symbol) or <\#U+20AC> (the Euro symbol)

The following special character tags were introduced in QuarkXPress 7.x, and will fail with “No such tag” on any other platform:

<code><\e>†</code>	Insert en space (half-em)
<code><_>†</code>	Insert Breaking em dash
<code><\5>†</code>	Insert 3-per-em space
<code><\\$>†</code>	Insert 4-per-em space
<code><\^>†</code>	Insert 6-per-em space
<code><\8>†</code>	Insert figure space
<code><\[>†</code>	Insert thin space
<code><\{>†</code>	Insert hair space
<code><\j>†</code>	Insert word joiner

Line Creation Tags

Create anchored line box (see page 42):

```
<&lb(px, py, flags, text align, line width, line color, line shade, line style, text outset, line endcaps, box name)>
```

Create unanchored line box (see page 42):

```
<&lbu(x, y, x2, y2, flags, runaround, line width, line color, line shade, line style, text outset, endcaps, box name, layer name)>
```

Master Page Tags (see page 59)

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

Macro Tags (see page 63)

Define a macro with a name and a body:

```
<&!(macroname, macrobody)>
```

Invoke a macro, substituting the given arguments *arg_i* for corresponding argument references in the macro's defined body:

```
<!macroname(arg1, arg2, arg3, ..., argn)>
```

When defining a macro, argument substitution references in *macrobody* are:

<code>!i</code>	<i>arg_i</i> , for <i>i</i> from 1 through 9 (single digit)
<code>!(i)</code>	<i>arg_i</i> , for any <i>i</i>
<code>!(i j)</code>	The <i>j</i> th character of <i>arg_i</i>
<code>!(i j:)</code>	The <i>j</i> th through the last character of <i>arg_i</i>
<code>!(i j:k)</code>	The <i>j</i> th through the <i>k</i> th character of <i>arg_i</i>

Paragraph Formatting Tags

<code><*L></code>	Align left (see page 74)
<code><*C></code>	Align center (see page 74)
<code><*R></code>	Align right (see page 74)
<code><*J></code>	Justify (see page 74)
<code><*F></code>	Force justify (see page 74)
<code><*h"H&J name"></code>	Apply hyphenation & justification, by name (see page 74)
<code><*d(chars, lines)></code>	Apply drop cap of number of characters/number of lines (see page 74)
<code><*knonoff></code>	Turn keep with next on (1) or off (0) (see page 74)
<code><*kt(A)></code>	Keep all lines together (see page 74)

`<*kt(start lines, end lines)` *Keep lines together at start and end (see page 74)*

`<*kt0>` *Keep no lines together (see page 74)*

Set basic paragraph parameters (see page 74):

`<*p(left indent, first indent, right indent, leading, space before, space after, lock to grid, language)>`

Set tabs (see page 74):

`<*t(position1, alignment1, "fill1", position2, alignment2, "fill2", ...)>`

Set no tabs (see page 74):

`<*t0>` or `<*t()>`

Set rule above (see page 74):

`<*ra(thickness, style, color, shade, [opacity,] from left, from right, offset)>`

Set no rule above (see page 74):

`<*ra0>`

Set rule below (see page 74):

`<*rb(thickness, style, "color", shade, [opacity,] from left, from right, offset)> ()`

Set no rule below (see page 74):

`<*rb0>`

Grouping Tag (see page 50)

`<&g(n1, n2, ...nm)>` *Group each n_i-most-recently created box*

Style Sheet Definition Tags (see page 30)

To define a paragraph style sheet:

`@name=[S] <tags>`

`@name=[S "based-on", "next", "char-style-sheet"] <tags>`

To define a character style sheet:

`@name=<tags>`

`@name=[S "", "", "", "based-on-char-style"] <tags>`

Style Sheet Application Tags (see page 30)

Apply named style sheet

PARAGRAPH	CHARACTER
<code>@name:</code>	<code><@name></code>

Apply Normal style sheet

<code>@\$:</code>	<code><@\$></code>
-------------------	--------------------------

Apply character style of current paragraph style

<code><@\$p></code>

Apply "No Style"

<code>@:</code>	<code><@></code>
-----------------	------------------------

Text Box Creation Tags (see page 33)

*Parameters for anchored and unanchored text boxes are very similar—when a parameter is unique, we've underlined it. Parameters whose names are shown in **bold** also have more advanced, sub-list forms which are summarized below. Parameters marked with a † have no tool defaults.*

Create anchored text box:

`<&tb(width†, height†, anchored alignment†, frame width, frame color, frame shade, frame style, background color, background shade, text outset, columns, gutter, text inset, baseline offset, baseline minimum, vertical alignment, interparagraph maximum†, box name†)>...Text to be placed in the box, typically containing other tags...<&te>`

Alternative form:

`<&tb2(width†height†box angle† box skew†flags† anchored alignment†, item runaroundframe widthframe colorframe shade frame style, background color, background shade, text outset, columns, gutter, text inset baseline offset, baseline minimum, vertical alignment, interparagraph maximum†, box name†)>...Text to be placed in the box, typically containing other tags...<&te>`

Create unanchored text box:

`<&tbu2(x† y†, width†height† box angle†, box skew†, flags†, item runaround, frame widthframe colorframe shadeframe style background color, background shade, text outset columns, gutter, text inset baseline offset, baseline minimum, vertical alignment, interparagraph maximum†, box`

`name†, layer name†)>...Text to be placed in the box, typically containing other tags...<&te>`

Text box parameter sub-lists

Both the anchored and unanchored text box tags have parameters that can be further defined as sub-lists. These include:

x ([see page 44](#))

(x offset, relative placement, relative box reference, relative box domain)

width ([see page 45](#))

(width, sizing specs, adjustment/margin, minimum)

height ([see page 46](#))

(height, sizing specs, adjustment/margin, minimum, leading adjustment)

frame width ([see page 48](#))

(frame width, corner diameter, corner type)

frame color ([see page 49](#))

(frame color, frame gap color)

frame shade ([see page 49](#))

(frame shade, frame gap shade, frame opacity, frame gap opacity)

background color ([see page 50](#))

(background color, blend color, blend style, blend angle, start position, center position, end position)

background shade ([see page 50](#))

(background shade, background opacity, blend shade, blend opacity)

text inset/outset ([see page 50](#))

(top, left, bottom, right)

Picture Box Creation Tags ([see page 37](#))

Parameters for anchored and unanchored picture boxes are very similar—when a parameter is unique, we've underlined it. Parameters whose names are shown

in **bold** also have more advanced, sub-list forms which are summarized below. Parameters marked with a † have no tool defaults.

Create anchored picture box:

```
<&pb(width†, height†, anchored alignment†, frame width,
frame color, frame shade, frame style, background color,
background shade, text outset, placement†, picture scale
x, picture scale y, picture offset x, picture offset y,
picture angle, picture skew, picture path name†, picture
type†, box name†)>¶
```

Alternative form:

```
<&pb2(width†, height†, box angle† box skew†flags† anchored
alignment†, item runaroundframe width, frame color, frame
shade, frame style, background color, background shade,
text outset, placement†, picture scale x, picture scale y,
picture offset x, picture offset y, picture angle, picture
skew, picture path name†, picture type†, box name†)>
```

Create unanchored picture box:

```
<&pbu2(x†, y†, width†, height†, box angle†, box skew†, flags†,
item runaround, frame widthframe colorframe shade frame
style, background color, background shade, text outset
placement†, picture scale x, picture scale y, picture
offset x, picture offset y, picture angle, picture skew,
picture path name†, picture type†, box name†, layer name†)>
```

Picture box parameter sub-lists

Both the anchored and unanchored picture box tags have parameters that can be further defined as sub-lists. These include:

x ([see page 44](#))

(x offset, relative placement, relative box reference)

width ([see page 45](#))

(width, sizing specs, adjustment/margin, minimum)

height ([see page 46](#))

(height, sizing specs, adjustment/margin, minimum, leading adjustment)

item runaround ([see page 48](#))

(frame width, corner diameter, corner type)

frame width ([see page 48](#))

(frame width, corner diameter, corner type)

frame color ([see page 49](#))

(frame color, frame gap color)

frame shade ([see page 49](#))

(frame shade, frame gap shade, frame opacity, frame gap opacity)

background color ([see page 50](#))

(background color, blend color, blend style, blend angle, start position, center position, end position)

background shade ([see page 50](#))

(background shade, background opacity, blend shade, blend opacity)

text outset ([see page 50](#))

(top, left, bottom, right)

None Box Creation Tags ([see page 41](#))

Parameters for none boxes are identical to text box and picture parameters of the same name. Please refer to those sections for more details. Parameters marked with a † have no tool defaults.

Create anchored none box:

```
<&nb(width†height† anchored alignment†, frame widthframe
colorframe shade frame style, background color, background
shade, text outset, box name†)>
```

Alternative form:

```
<&nb2(width†height† box angle† box skew†flags† anchored
alignment†, item runaroundframe widthframe colorframe shade
frame style, background color, background shade, text
outset, box name†)>
```

Create unanchored picture box:

```
<&nbu2(x† y†, width†height† box angle†, box skew†, flags†,
item runaround, frame widthframe colorframe shadeframe
style background color, background shade, text outset, box
name†, layer name†)>
```

Table Creation Tags ([see page 37](#))

Parameters for anchored and unanchored table boxes are very similar—when a parameter is unique, we've underlined it. Parameters whose names are shown in **bold** also have more advanced, sub-list forms which are summarized below. Parameters marked with a † have no tool defaults.

Create anchored table box:

```
<&ts(width†, height†, number of columns, number of rows,
column widths, flags, anchored alignment†, frame width,
frame color, frame shade, frame style, background color,
background shade, text outset, gridline width, gridline
style, gridline color, gridline shade, [gridline opacity,]
gridgap color, gridgap shade, [gridgap opacity,] tab
order, link order, box name†)>...ROWS...<&tse>
```

Create unanchored table box:

```
<&tsu(x†, y†, width†, height†, number of columns, number of
rows, column widths, table angle†, flags, item runaround†,
frame width, frame color, frame shade, frame style,
background color, background shade, text outset, gridline
width, gridline style, gridline color, gridline shade,
[gridline opacity,] gridgap color, gridgap shade, [gridgap
opacity,] tab order, link order, box name†, layer name†)>...
ROWS...<&tse>
```

Table box parameter sub-lists

Both the anchored and unanchored table box tags have parameters that can be further defined as sub-lists. These include:

x ([see page 44](#))

(x offset, relative placement, relative box reference, relative box domain)

width ([see page 53](#))

(width 1, width 2, width 3, ...)

height ([see page 53](#))

(height, sizing specs, adjustment/margin, minimum, leading adjustment)

column widths ([see page 53](#))

(column 1 width, column 2 width, column 3 width, ...)

frame width ([see page 48](#))

(frame width, corner diameter, corner type)

frame color ([see page 49](#))

(frame color, frame gap color)

frame shade ([see page 49](#))

(frame shade, frame gap shade, frame opacity, frame gap opacity)

background color ([see page 50](#))

(background color, blend color, blend style, blend angle, start position, center position, end position)

background shade ([see page 50](#))

(frame width, corner diameter, corner type)

text inset/outset ([see page 50](#))

(top, left, bottom, right)

gridline/gridgap attributes ([see page 55](#))

(top, left, bottom, right, all interior horizontal, all interior vertical)

Table row and cell tags

Create table row ([see page 56](#)):

```
<&trs(type, height, background color, background shade,
gridline width, gridline style, gridline color, gridline
shade, [gridline opacity,] gridgap color, gridgap shade [,
gridgap opacity])>...cells...<&tre>
```

Create table picture cell ([see page 57](#)):

```
<&tcp(width, height, horiz span, vert span, flags,
background color, background shade, placement, picture
scale x, picture scale y, picture offset x, picture offset
y, picture angle, picture skew, picture path name, picture
type)>
```

Create table text cell ([see page 57](#)):

```
<&tcs(width, height, horizontal span, vertical span, flags,
background color, background shade, text angle, text skew,
text inset, baseline offset, baseline minimum, vertical
alignment, interparagraph maximum)>...text contents...<&tce>
```

Both the text and picture cell tags have parameters that can be further defined as sub-lists. These are the same as the sub-lists for text and picture box parameters.

Translation Table Invocation Tags ([see page 52](#))

```
<&tt2"table name">
```

start using the specified translation table

```
<&tt2w"table name">
```

same as &tt2 tag, but don't provoke an error if the called-for translation table file isn't found

Xcatalog/InCatalog Link Creation Tags ([see page 70](#))

Start a text selection link:

```
<&Cs>
```

End a text selection link, with the link information:

```
<&Ce(dd, flags, key, keytype, field, subfield, picture position,
price style)>
```

Create a link on a relatively-referenced box:

```
<&Cb(relative box reference, dd, flags, key, key type, field,
subfield, picture position, price style)>
```

Appendix B

Error Handling

This section explains each of the Xtags error messages. Note that these messages appear only when error reporting is enabled. In-line error reports, when requested, are always inserted in the character style settings of the original selection or insertion point at the start of the input. In-line error reports add important error information, letting you know which tag and which parameter—for list-style tags—is involved.

Parameters Out of Range

When a parameter given to a tag is below its minimum acceptable value or above its maximum value, that parameter is silently set to the minimum or maximum value itself, respectively (because this is how the XPress Tags filter works).

Error Alerts

Xtags encountered *n* errors during import (the text «Xtags error: description» is left at each point of error). [*multiple errors encountered*]

This alert, given after an import is finished only when the Report Errors option was selected, tells you how many errors were detected and reported. As noted in the message itself, a terse error report (elaborated below) is left in the resulting text stream at each point of error.

**Stopped at the demonstration copy limit of 50 paragraphs.
Stopped at the demonstration copy limit of 50 paragraphs, because you're over the max # user limit for this copy of Xtags.**

The first error is given when you're trying to import or export more than 50 paragraphs with the demo version of Xtags. The second is given when you're trying to do the same, but with a regular version of Xtags that's been turned (silently) into a demonstration copy because it detected more than the number of licensed copies running on the network when it started up (you can see this reported in the **Utilities->Xtags->About...** dialog). To fix this problem, reduce the number of simultaneous copies, by disabling one or more copies

on other systems and restarting QuarkXPress on those systems. (Or, consider upgrading to the next level of *n*-pack license—we have generous discounts for same.)

Xtags can't find a translation table file named in a &tt/&tt2 tag (system error code: ____).

Xtags couldn't find translation table by name, because no such file exists in the folders/directories Xtags searched (see the &tt2 tag description for details on translation table lookup). Typical MacOS system error codes are -43 for file (or folder) not found, -120 for folder not found, -47 for file is "busy" (open for writing elsewhere), etc. Similar codes will be encountered under Windows.

No automatic text flow on newly-applied master page: Xtags can't continue.

You've applied a master page that has no automatic text flow text boxes on it. Xtags has to give up, because it can't place any more text.

Error Reports

Each of these errors is inserted in the text stream, at the point of error, in the form «Xtags error: description».

Box is too large to fit on spread.

The unanchored box as specified won't fit on the spread in one dimension or in both dimensions, even after it was moved as far up and/or as far left as possible.

Box placement failed.

Xtags can't create an unanchored box failed for some reason (probably lack of memory or an internal error).

Can't add style: no memory.

Defining a new style (@...=...) failed because there's not enough memory to hold the temporary new style.

Can't add style to document.

For some reason, defining a new style (@...=...) failed. Most likely, the maximum number of styles per document has been exceeded under QuarkXPress.

Can't anchor box.

Xtags can't insert a newly-created anchored box in the text stream for some reason (&tb or &pb tag), probably due to lack of memory or an internal error.

Can't anchor box inside anchored box.

You're trying to use a `&tb` or `&pb` tag inside a `&tb/&te` tag pair, and QuarkXPress doesn't support anchoring boxes inside anchored text boxes.

Can't apply master page/spread.

The master page/spread application (`&m/&mp/&mf/&mpf` tag) failed for some reason, even though a valid master page is being applied. (Probably an internal error.)

Can't create box.

Xtags can't create a temporary box during anchored box creation (`&tb` or `&pb` tag), probably due to lack of memory or an internal error.

Can't find picture file.

Xtags can't find the picture file as named in a `&pb/&pbu2` tag.

Can't fit: general problems.

A generic error for shrink-to-fit failures (e.g., out of memory, XPress failure, etc.).

Can't fit: picture is empty.

A shrink-to-fit picture box size specification was given, but no picture is present.

Can't fit: picture too large.

A shrink-to-fit picture box would have to expand, not shrink, to accommodate its current contents.

Can't fit: text box not visible.

A shrink-to-fit text box is in overflowed text and can't be shrunk to fit.

Can't fit: text is empty.

A shrink-to-fit text box is empty, and it can't be shrunk to zero height.

Can't fit: too complex.

A shrink-to-fit text or picture box exceeds Xtags' current implementation limitations (e.g., you're trying to shrink-to-fit a text box with multiple columns).

Can't fit: too much text.

A shrink-to-fit text box would have to expand, not shrink, to accommodate its current contents.

Malformed number.

A numeric parameter is malformed in some way: it contains non-numeric characters that aren't valid unit designators (e.g., `<z(3pts)>` where `pts` should have been `pt`), or embedded spaces (e.g., `<*p(1 2, ...)>`), etc.

Can't group: box is already grouped.

A grouping tag (`&g`) references a box that already belongs to a group.

Can't group: boxes aren't on same spread.

A grouping tag (`&g`) references two or more boxes that don't fall on the same spread.

Can't import picture.

Xtags can't import the named picture in a `&pb` tag, most likely because the picture isn't one that QuarkXPress knows how to import.

Can't place box: text overflow.

Can't place an unanchored box (`&tbu2/&pbu2` tags) without (x, y) coordinates because the current insertion point is in overflow text, and thus Xtags can't compute a place for it.

Hidden text open/close mismatch.

A run of hidden text was active when either another hidden text tag (e.g. `<A()>` or `<&Cs>`) was encountered or a box transition (e.g. `<&tb()>` or `<&te>`) occurred.

List element too long.

More than 255 characters are being used in a list element.

Macro body too long.

A macro's definition body won't fit in the prescribed limit (4096 characters, currently).

Macro expansion too long.

A macro being expanded at invocation time would be too long for the available space (4096 characters, currently).

Macro name too long.

More than 255 characters are being used in a macro name.

Malformed definition prefix.

Something is wrong with a style definition prefix (the code in brackets after the equal sign of a style definition): either the required `S` is missing, an end of line was found before the end of the based-on style name was seen, or there is no closing bracket, e.g., respectively:

```
@style=["Normal"]<...>
@style=[S"Normal
@style=[S"Normal"<...>
```

Malformed macro argument reference.

A macro argument reference is malformed (e.g., `!(1 a)` where the non-numeric `a` is used in an argument reference).

Malformed tab alignment spec.

One of the tab alignment parameters in a `*t` tag is neither a valid string value (to align on the specified ASCII non-control-character) nor a valid number from 0 to 3.

Malformed tab leader spec.

One of the tab leader parameters in a `*t` tag is neither a single character "X" nor a specification of the form "1XX" or "2XY".

Malformed tag.

This is a catch-all error for general structural problems with tags, such as a missing right parenthesis in a list tag (e.g., `<*p(1,2)>`), a parameter to a list tag that isn't a list or, when appropriate, a \$ (e.g., `<*t1>` or `<&tb1>`), a parameter that isn't one of the permitted choices (e.g., `<&tb(72, 144, C)>`, where **C** should be either **A** or **B**), or a continuation colon in a tag followed by an end-of-tag instead of a return (e.g., `<z10:>`).

This error is often the result of an incorrect number of commas for defaulted parameters in the text and picture box creation tags.

Missing font.

The font name given to the `f` tag is known in the context of this document, but currently not available in the current environment.

Nested macro invocation unsupported.

A nested macro invocation was found and is not supported by Xtags, currently (e.g., `<!m1("!m2(a)", b)>`).

No default value available (for \$).

A \$ was used where a default value isn't available (e.g., in a tab indent specification, or a rule-above or rule-below width specification, where there's no rule above or below setting defined for the current style).

No such based-on style.

The based-on style in a style definition prefix is not a known style in the current document (it may be misspelled).

No such color.

The parameter given to a tag expecting the name of a document color is not a valid color name, nor is it a valid color abbreviation (e.g., `<c"Whote">`, where **White** was meant).

No such H&J.

The H&J specification name given to the `*h` tag isn't known in the current document.

No such frame.

The frame index given as a parameter to the `&tb`, `&tbu2`, `&pb` or `&pbu2` tag is beyond the known frame indices in the current QuarkXPress preferences, or the frame name given wasn't found.

No such font.

The font name given in the `f` tag currently isn't known. Be sure that the font name is spelled out in full (not abbreviated, as is possible with ordinary XPress Tags).

No such language.

The language name given to the `<*p(>` tag isn't known.

No such line style.

The line style parameter in a `*ra` or `*rb` tag is not one of the valid styles. (See the `*ra/*rb` tag documentation for the complete list of valid styles.)

No such macro.

A macro was invoked that hasn't been defined. Check your spelling.

No such master page.

The master page identifier given to `&m/&mp/&mf/&mpf` is either out of range (if an index) or is not a valid master page name (if a string). For example, if you have three master pages named **Masthead**, **Left**, and **Right** in a document, then the only valid master page identifiers for the `&m/&mf` tags when importing into this document are "Masthead", "Left", "Right", **1**, **2**, **3**, as well as the full prefixed master page names. Anything else (e.g., **0** or **4** or "Master A") will trigger this error report.

No such next style.

The next-style given in a style definition is not a known style in the current document (it may be misspelled).

No such previous box.

A relative box reference is within a valid range, but refers to a box we haven't created (yet).

No such tag.

A tag sequence contains a tag that isn't understood. The most common cause of this error is a stray tag-begin (left angle bracket) in text (e.g., `when x < y`), which causes Xtags to start interpreting the following text as XPress Tags.

Not enough memory for translations.

Xtags is attempting to expand its translation table and can't allocate enough memory to do so.

Translation file not found.

Xtags cannot find the specified translation file. See chapter 10 for the folder search order.

Not enough memory for macro definition.

Xtags is attempting to add a macro definition and can't allocate enough memory to do so.

Not in main text flow.

You've tried to group some boxes or apply a master page/spread inside a nested text box tag (e.g., `<&tb(...)&g(1,2)&te>`).

Number too long.

More than 255 digits are being used to form a number.

Paragraph element in character style definition.

A paragraph attribute tag (i.e. any of the `<*>` tags, like `<*p(>` and `<*J>`) was encountered while defining a character style.

Relative box reference out of range.

A relative box reference (in a `&g` tag parameter or in a relative box placement specification) is less than 1 or greater than the maximum number of boxes that can be referenced relatively (99, currently).

String too long.

More than 255 characters are being used to form a string.

Style name too long.

More than 64 characters are being used in a style name.

Sub-list element too long.

More than 255 characters are being used in a sub-list element.

Text boxes nested too deep.

You've nested text box creation beyond the limit (8).

Too many tabs.

More than 20 tabs are being defined in a `*t` tag. (Obsolete.)

Unexpected end of input.

Xtags is generally very tolerant of where it finds the end of file (anywhere outside of a tag sequence is fine, and most anywhere inside a tag sequence is acceptable as well, for compatibility with XPress Tags), but some situations cry out for intolerance. E.g., if the tag fragment `<*p(` is the very last set of characters in the input file, then Xtags has to assume that something is awry, and presents you with this complaint.

Too many tabs.

More than 20 tabs are being defined in a `*t` tag. (Obsolete.)

Unexpected end of line.

Normally, carriage-returns aren't acceptable in the middle of a tag sequence, but Xtags has found one; if you want to continue a tag on a new line, precede the carriage-return with a colon to indicate continuation.

Unmatched &te.

A `&te` tag has appeared without a preceding matching `&tb`.

Value out of range.

A numeric parameter is out of range for the value in question (e.g., a paragraph right indent value is less than the negative of its left indent value).

Invalid background blend specified.

An unrecognized blend name was specified for a box (*see page 50 for valid options*).

Table not started.

A table tag was encountered outside of a table start (`&ts` or `&tsu`)/ table end (`&tse`) tag pair.

Table row not started.

A table row end (`&tre`) or table cell start (`&tcp` or `&tcs`) tag was encountered without an opening table row start (`&trs`).

Table cell not started.

A table cell end (`&tce`) tag was encountered without a matching table cell start (`&tcp` or `&tcs`).

Attempts were made to access a column beyond the end of the table.

Table columns exhausted.**Table rows exhausted.**

Attempts were made to access a column or row beyond the end of the table.

Can't add table column.**Can't add table row.**

The table has grown past the maximum size allowed.

Can't create layer.

The specified layer did not exist, and Xtags ran into an error when trying to create it.

Can't set the box layer

Xtags encountered an error while attempting to move a newly-created box to its named layer.

AppleEvent Errors (MacOS only)

These errors are returned as the error number ('**errn**' parameter in AppleEvent terminology) and error string ('**errs**' parameter) in the reply to a failing **get text with Xtags** event, which will need to be caught with an **on error** clause in an Applescript **try** construct. The error strings are as follows. (The text of interactive error alerts (described above) may also be returned as error messages.)

Bad file specification given.

Extracting the given file specification failed.

Can't convert file spec to internal form.

Converting the given file specification to Xtags' internal form failed.

Current box is not text box.

The current box in the current document isn't a text box, so no import is possible.

No current box.

There is no current box in the current document, so no import is possible.

No current document.

There is no current document, so no import is possible.

No source specified.

No **from file** or **from string** parameter was given.

Source is neither string nor file.

The given **from** source must either be a string or something coerceable to a file specification.

Toolbox or XPress error.

Generic failure (no specifics known).