Xtags

Version 3.x/4.x

Em Software, Inc.

Xtags™ 3.x/4.x User's Guide

Em Software, Inc.
January, 1999

Keeping in Touch with Em via the Web

Em maintains a World-Wide Web (Internet) site at www.emsoftware.com, with up-to-date information about Xtags and its other products. We will provide news of major updates, minor version software updaters, free software, on-line tips and techniques to augment this manual, etc., at this site, so we encourage you to check it every so often.

Technical Support

If you have problems using Xtags, please first consult the "Error Handling" chapter in this manual to find many common problems and their solutions—your answer is very likely to be found there, and you can save yourself a lot of trouble with just a little reading. For additional information, consult the frequently-asked questions and other usage hints on the support pages of Em's web site at www.emsoftware.com/support.html.

If neither of these sources help, feel free to contact us directly, ideally via fax or email. (We'll ask you for details via fax or email anyway, if you call.)

For the most effective support via fax or postal mail, please send (a) a printed listing of your Xtags input or output tagged text, (b) the relevant printed pages of the document in question, and (c) a cover sheet describing the exact problem and how to replicate it. For the most effective support via email, please send a self-extracting StuffIt, WinZip or other archive containing (a) the XPress document (or template) involved, (b) your input or output tagged text file, and (c) a "read me" text file or QuarkXPress document describing the exact problem and how to replicate it.

Contact Information

Em Software, Inc.
503 Belleview Blvd.
Steubenville, Ohio 43952 USA
vox 740 284 1010, ***fax*** 740 284 1210
web www.emsoftware.com
email support@emsoftware.com *or* info@emsoftware.com

Copyright and Trademark Information

Credits

This manual was written by Æleen Frisch, Exponential Consulting, Northford, Connecticut. The software was conceived and implemented by Chris Ryland, Em Software (AMDG). The Windows port was performed by Chris Roueche. The Em logo (on the manual cover) was created by A&M Design, Madison, Connecticut.

Production Notes

This manual was composed using QuarkXPress 4.03 for the Macintosh and Windows. Body text is set in 10/11 New Century Schoolbook type; headings are in Helvetica. The manual masters were typeset and offset printed in the U.S.A. at CopyMasters, Taunton, Massachusetts.

Version Information

Manual version 3.0/4.0, corresponds to Xtags version 3.02/4.02, January, 1999.

Table of Contents

1

Introduction to Xtags

Xtags™ provides an enhanced version of the XPress Tags language supported by QuarkXPress.® Both XPress Tags and Xtags allow you to import plain text files into QuarkXPress as fully formatted text, using special formatting codes embedded in the text file; you can also export formatted text within a QuarkXPress document via both XPress Tags and Xtags. Except as noted in this manual, all XPress Tags constructs are supported by Xtags.

Xtags adds several major features to the basic XPress Tags language, including the ability to:

- create and fill both in-line anchored and unanchored (out-of-line) text and picture boxes, including shrinking boxes to fit their contents;
- group created text and/or picture boxes;
- apply master pages to the current page or spread;
- translate user-defined tags into standard Xtags constructs (or into anything else);
- use macros to express complex, parametrized tag sequences as shorthand (for example, fractions).

Xtags is an XTensions™ to QuarkXPress, automatically becoming part of the QuarkXPress publishing system once it is installed. Xtags is invoked via the **File** and **Edit** menus, and imports tagged text into normal QuarkXPress publications, as well as exporting tagged text. It is not a stand-alone program, separate from QuarkXPress, and no special additional files are required. (See the QuarkXPress manual for general information about Xtensions.)

System Requirements

In order to run Xtags, you will need a Macintosh or an Intel Windows-based computer running QuarkXPress version 3.3 or later. The computer will need sufficient memory and hard disk space to support QuarkXPress (consult the QuarkXPress documentation for details).

MacOS-based computers may run either System 7 or System 8. Windows 95, 98 and Windows NT 4.0 are supported on Intel systems, to the extent that QuarkXPress itself supports each operating system. (While it's likely that Xtags 3.x will run under Windows 3.1 or 3.11, we don't officially support these operating systems; in particular, we use long file names on the distribution CD which are difficult to navigate under these older systems.)

We split the Xtags product into two distinct but parallel XTensions: one that runs natively under QuarkXPress 3.3 to 3.32 (Xtags 3.x), and one that runs natively under QuarkXPress 4.0 and later (Xtags 4.x). (We advanced and split the major version number from 2.x to 3.x and 4.x to make it clear which Xtags works with with QuarkXPress.)

All Xtags releases will come in these version 3.x/4.x matching pairs (e.g., 3.02 and 4.02 are a pair) as long as QuarkXPress 3.3 is a viable product. The feature sets of any two matched versions should be identical, except where the Xtags 4.x product takes advantage of new QuarkXPress 4.0 features (such as character styles). Our intent with these 3.x/4.x pairs was to make sure you always have the appropriate version of Xtags at your fingertips, no matter which version of QuarkXPress you decide to use.

We heartily recommend that you run QuarkXPress 3.32r5 (on MacOS) or 3.32r3 (on Windows) if you're running any other QuarkXPress 3.x version, since these final versions are quite stable, in our experience and in others'.

What You Need to Know …

About Your Computer

You should be familiar with basic Macintosh 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).

About QuarkXPress

You should be comfortable with basic QuarkXPress tasks. You 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.
- Define, apply, modify and delete style sheets.
- Import text and graphics into QuarkXPress.
- Set up and assign master pages.
- Save and print publications.

New Features in Xtags 3.x/4.x

Xtags 3.x/4.x is a major release, incorporating a slew of major features, smaller improvements, and bug fixes. Xtags 3.x/4.x (all new features are explained later in this document in more detail):

- adds full support for all new and changed QuarkXPress 4.0 XPress Tags (implementing version 2.x of the tag language, with all its quirks). [Xtags 4.x only]
- extends the *frame color* parameter of a box-creation tag to a tuple (*frame fg color*, *frame bg colo*r) to give you control over both the foreground and background colors in a frame, and extends the *frame shade* parameter in a box-creation tag to a tuple (*frame fg shade*, *frame bg shade*) to give you control over both the foreground and background shades of a box's frame. [Xtags 4.x only]
- supports a *frame style* specification in a box creation tag of either a frame name (as given in the frame style menu)—e.g., "Single"—or a resource identifier, instead of just the frame style menu index of version 3.x. [Xtags 4.x only]
- adds a relative-sizing facility, for specifying a box's width or height in terms of another, recently-created box's width or height, plus or minus an optional adjustment.
- adds a horizontal fit-to-box sizing option (to be used in conjunction with the vertical size-to-fit option), which allows you to specify that a picture should be scaled to fit its containing box horizontally (plus or minus some margin), and then this resulting scale used as the picture's vertical scale. If you also use the (existing) vertical size-to-fit option, then the box will be fit to the resulting picture vertically as well.
- adds a fix-paragraph-leading facility for anchored boxes which will set a newly-created box's containing paragraph's leading to the final height of the box (including text outset), plus or minus an optional amount.
- adds a minimum-size specification facility for auto-sizing of any box in either dimension, such that you can specify the minimum dimension of any auto-sized or relatively-sized box.
- supports the specification of corner type and diameter in unanchored box creation tags.
- supports multiple text outsets (for the item runaround case) on unanchored text or picture boxes.
- supports multiple text insets on anchored or unanchored text boxes.

- supports AppleEvent box names as a new final parameter to all picture and text box tags (anchored and unanchored) on both input and output (platform-independently, though the names can only be used on the MacOS platform).

- supports creation of Xcatalog 3.x/4.x text selection and box links with three new tags, **&Cs**, **&Ce**(...) and **&Cb**(...).

- supports recursive groups on input and output.

- adds the new AppleEvent **save text with Xtags**.

- adds support for the AppleEvent form **get text with Xtags from** *string*.

- adds the ability to **Save Xtags Text...** and **Copy Xtags Text** in mover tool mode, including supporting groups (but not a non-grouped multiple selection) and outputting the box creation tag as well as the content tags for text boxes.

- adds important further error information to in-line error messages (which tag and which parameter—for list-style tags—is involved).

- supports any number of parameters to the **&!** macro-definition tag, appending the second through the last into one long body, up to 4,096 characters worth (increased from 255 bytes).

- supports the auto-image (a) run-around type on input and output, along with the item (i) and manual (m) run-around types.

- now correctly outputs **A** or **M** for auto image or manual image runaround types, respectively, on box output (the latter will cause a malformed tag error on input).

- adds a new flag parameter possibility **L** to **&tbu2()** and **&pbu2()** tags to lock the newly-created box (as if you had used the **Item→Lock** menu item on it).

- adds new error messages for number, string, list element, sub list element, style and macro name too long (> 255 characters in length), and the offending elements are now parsed properly so Xtags doesn't get confused.

- adds support for quotes within strings using the backslash (****) escape character, for both input and output.

- supports full (including prefix) master page/spread names in master page/spread application tags, as well as the (non-prefixed) shorter version.

For example, now the full name “A-Master A” will work as well as the shorter name “Master A”.

Incompatible Change in Xtags 3.x/4.x for Windows

To keep up with changes in the XPress Tags 3.3x filter, we had to introduce a change which will break any existing Windows pathnames (i.e., more than just a filename) in **&pb()** and **&pbu2()** tags: the backslash character (****) is now the escape character inside strings, and therefore ***must*** be doubled in pathname strings to obtain a single backslash.

For example, the Windows pathname "Pictures\Capital.tif" must now be specified as "Pictures\\Capital.tif".

We always try to keep existing Xtags input files working in updated versions, but we didn’t have a choice in this case because of Quark's changes.

About This Manual

The next two sections of the manual describe how to install and use Xtags. The main part of the manual describes the Xtags language, including most “vanilla” XPress Tags constructs; it is divided into numbered sections which group tags of similar scope and purpose. The final section of the manual contains a detailed description of Xtags’ error handling.

Platform-Specific Issues

Xtags 3.x/4.x has been designed to work as similarly as possible on all of its supported computer and operating systems. Differences between versions will be noted as appropriate. In selecting illustrations of dialogs, we will generally show Macintosh versions. Note, however, that the dialogs look very much alike under both operating systems, except for graphic details. In this manual, we will consistently refer to directories and subdirectories as “folders,” a term which is used in both Macintosh and Windows parlance.

Typographical Conventions

We use alternate typefaces in this manual to distinguish certain types of items within the text. Often, Xtags and XPress Tags will be set off in separate paragraphs set in boldface Helvetica type, like this:

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

When we need to talk about tag elements within a paragraph, we’ll again set them in boldface Helvetica type. For example, the previous example illustrates the **<*L>** tag.

When we are discussing tag syntax, we will use italic Helvetica 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:

<f"*font-name*">

Here, *font-name* is a placeholder for a font name that you would provide when you actually used this tag. Notice that we set such parameters in italics when we use them within regular text as well.

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 QuarkXPress and Xdata menu items and dialog box fields are set in boldface type, as in "select **Open...** from the **File** menu" and "click the **Cancel** button to change your mind." We use the following syntax to represent paths through a series of nested menus: the form **Utilities→Xtags→Preferences...** means to select the **Xtags** item from the **Utilities** menu in QuarkXPress and then to select the **Preferences...** item from the **Xtags** slideoff submenu.

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."

2

Installing Xtags

This section describes the steps needed to install Xtags.

Xtags Disk Contents

Xtags comes either on a cross-platform (Windows and MacOS) CD with all other Em Software "Work Smart" products (Xdata and Xcatalog as well)—with unlocking codes (serial numbers) for what you've purchased—or in an electronic distribution.

The Xtags contents can be found in the Xtags folder, as follows:

Xtags 3.*xy* (Macintosh); **Xtags3***xy***.XXT** (Windows)
The Xtags XTensions (e.g., Xtags 3.01), to be used only with QuarkXPress 3.3, 3.31 or 3.32, where *x* is the minor version and *y* is the fix level (which may be blank).

Xtags 4.*xy* (Macintosh); **Xtags 4.***xy***.XNT** (Windows)
The Xtags XTensions (e.g., Xtags 4.01), to be used only with QuarkXPress 4.z, where *x* is the minor version and *y* is the fix level (which may be blank).

Read Me (Macintosh); **ReadMe.TXT** (Windows)
A text file (readable with any editor) containing last-minute notes, known problems with Xtags and QuarkXPress, any system or XTensions conflicts, etc. We highly recommend you read this to avoid any nasty surprises.

Samples (Windows and Macintosh)
A folder containing demonstration files, including its own **Read Me** file explaining what each sample does and how to use it (so that the demonstration and release distributions can be identical).

There may be additional files in the Xtags folder on CD. The **Read Me** file will describe any such files.

Installing the Xtags XTensions

Installing the Xtags Xtensions is simple: copy the appropriate Xtags XTensions file for your version of QuarkXPress from the distribution folder to the QuarkXPress application folder's **XTension** subfolder. You will need to (re)start QuarkXPress in order to make Xtags available. Note that if you are using both QuarkXPress 3.3x and 4.x, you will need to copy the appropriate version of Xtags to the **XTension** subfolder for each copy of QuarkXPress.

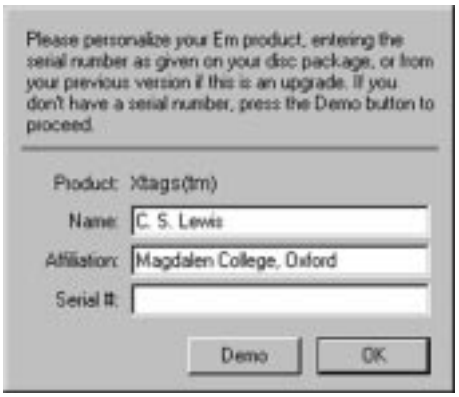
XTensions and Memory Consumption

All XTensions to QuarkXPress consume some portion of the available memory. If memory is limited on your system, then you should place only those XTensions into the QuarkXPress **XTension** folder that you use frequently. XTensions that you use on an irregular basis may be stored in a separate folder inside the QuarkXPress folder, often named **XTension Disabled**. When you want to use one of these XTensions, move it back into the **XTension** folder before starting QuarkXPress; when you are finished with it for awhile, move it back to the unused XTensions storage folder after quitting QuarkXPress.

Owners of multi-pack versions of Xtags should be aware that moving an XTensions to the **XTension Disabled** folder, either manually or via the QuarkXPress XTension Manager) does not deactivate it from a licensing point of view. It will still consume one serial number. In order to release its slot on the network, you must move the XTensions file to another folder. We recommend creating an **XTension Really Disabled** subfolder within the QuarkXPress folder for such purposes.

Personalizing Your Copy of Xtags

The first time that you start up QuarkXPress after installing Xtags, you will be prompted to fill in your name, your organizational affiliation and your Xtags serial number:



You'll find your Xtags serial number printed on labels placed on the Xtags disc envelope, on the registration card, and on the shrink-wrap license disc assem-

bly. Or, it may be provided over the phone, by email or via fax if you're registering a previous demo version. If you are upgrading from Xtags 2.x, simply use the same serial number that came with that version.

Be sure to record your Xtags serial number somewhere safely and permanently. We suggest writing it on the inside cover of your Xtags manual.

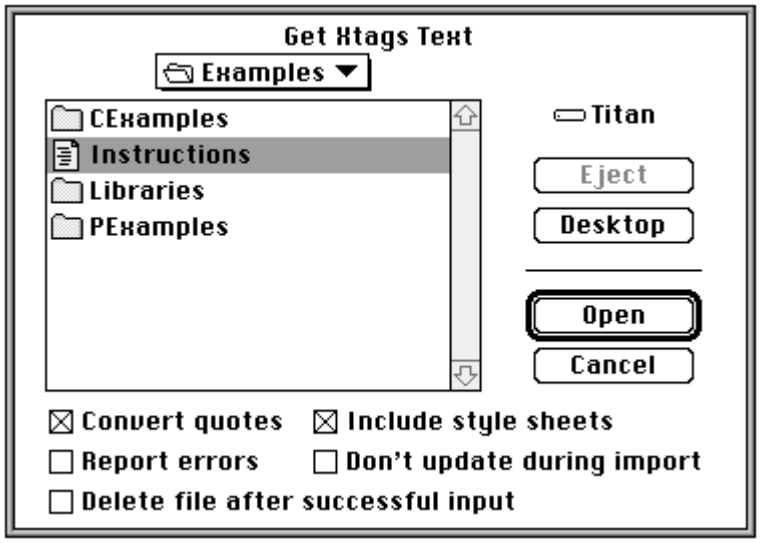
Note that, under QuarkXPress 4.x on either platform, the serial number is stored in a file called **Xtags.Reg** in the QuarkXPress folder. If you move Xtags to another machine, you should also move this file unless you don't mind re-entering your serial number. And, if you install later (free) upgrades for Xtags, you won't have to re-enter the serial number after installing Xtags, as it'll be read from this file. This information may be viewed at any time by selecting the **Utilities→Xtags→About...** menu item within QuarkXPress.

3

Importing and Exporting Text with Xtags

Normally, ASCII text files containing XPress Tags are imported using **Get Text...** on the **File** menu. When the **Include Style Sheets** box is checked, the tags in the incoming text are interpreted as it is placed into the QuarkXPress document, resulting in typeset text with the specified character and paragraph formatting.

The import process using Xtags is very similar. Xtags adds two additional items to the QuarkXPress **File** menu. The first of these—**Get Text with Xtags...**—enables you to import text styled with Xtags. Selecting this item opens the **Get Xtags Text** dialog:



This dialog is a modified version of the normal QuarkXPress **Get Text** dialog. Once you have selected the options (discussed in a moment) and file you want to import, press the **Open** button (or double click on the desired file name) to begin the import process.

Error Handling

As the import finishes, if Xtags encountered any errors and you have selected the Report Errors option, it will alert you with an error dialog that tells you exactly how many errors it found. 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 (as applicable). These messages and their meanings are documented in the final section of this manual, “Error Handling.”

Xtags Import Options

The check boxes in the **Get Xtags Text** dialog have the following meanings:

Convert quotes
Tells Xtags to automatically change any straight quote (" or ') to the appropriate printer’s quote, depending on its context: "sample" becomes “sample” and 'sample' becomes ‘sample’. (This operates the same way as the similarly-named option in **Get Text...**.)

Include style sheets
Tells Xtags to interpret tags as formatting instructions. Must be checked for normal Xtags operation (this is also true when importing normal XPress Tags text via **Get Text**).

Report errors Tells Xtags to insert a brief error message into the incoming text 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 (unlike XPress Tags, which stops importing at the first error, with no error indication), so if you don’t select this option, Xtags will do its best in the face of errors and never tell you a thing.

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. On the Macintosh, this is accomplished by mov-

ing the file to the trash can, and under Windows, the file is moved to the recycle bin.

These options may be preset in the **Xtags Preferences** dialog (discussed later).

Saving Text with Xtags

The **Save Text with Xtags...** menu item on the QuarkXPress **File** menu may be used to save text styled with Xtags (suitable for subsequent importing). This option saves any selected text to the file you specify. The default filename 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.

If no text is selected when you select **File→Save Text with Xtags...**, then the entire current story will be saved.

If an unanchored box is selected, then the tags necessary to recreate that unanchored box will be saved.

If a group (not a multiple selection) is selected, then the tags necessary to recreate that group will be saved.

Any anchored boxes included among the selected text will produce the tag 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 QuarkXPress **Edit** menu:

Copy Xtags Text

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**. This item is enabled only when there is a text selection or picture box selected in content mode, or a group selected in item tool mode. The save/export settings in the **Xtags Preferences** dialog are in effect.

Paste Xtags Text

Import text on the system clipboard, interpreting it as Xtags input. This item is enabled only when there is text present on the system clipboard. The import settings in the **Xtags Preferences** dialog are in effect.

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 or a sample picture box (or group) as you desire, 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 QuarkXPress **Edit→Paste** command.

Similarly, the **Paste Xtags Text** facility may be used to quickly preview the results of some tagged text you are creating. Select the 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 then choose **Edit→Paste Xtags Text**.

Xtags Preferences

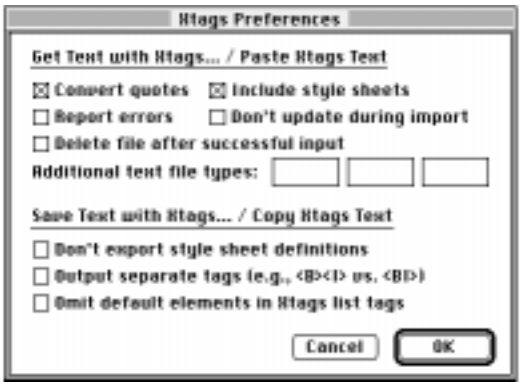
Xtags adds a new item to the **Utilities** menu: **Xtags**. Selecting this item reveals a submenu, containing two items:

Preferences... Allows you to specify the global default setting for each of the check box options in the **Get Xtags Text** dialog (these settings also apply to **Paste with Xtags** operations), default settings for save and copy Xtags text operations, and additional extensions used for text files on your computer system. Note that these default settings persist across QuarkXPress restarts.

About... Displays the version number, serial number, and other information about this copy of Xtags.

The Xtags Preferences Dialog

The **Xtags Preferences** dialog allows you to specify a variety of Xtags defaults. It is divided into two areas, controlling importing and exporting text with Xtags, respectively:



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 (discussed earlier).

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 on the Macintosh and are designated by file extensions of one to three (usually three) characters under Windows.

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
: 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 **<B><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, the tag on the left illustrates the output format for a 6-point 50% blue rule above a paragraph in the default output mode, and the example on the right illustrates the output that would result when this box is checked:

UNCHECKED	CHECKED
<*ra(6,0,"Blue",50,0,0,0)>	**<*ra(6,,"Blue",50,,,)>**

Automating Document Building with AppleScripts

On Macintosh computer systems running System 7 or later, Xtags supports AppleEvent scripting for automating document building via the AppleScript scripting language. This section assumes some very basic familiarity with AppleEvents and scripting, such as why it's useful and how it works at some very high level.

Xtags adds the get text with Xtags event to the built-in XPress events. The basic structure of the Xtags import statement in AppleScript is:

```
get text with Xtags from source
        [ with / without error reporting ]
        [ with / without document updating ]
        [ with / without quote conversion ]
```

You must first have a document open and a text box selected before invoking Xtags in this fashion from a script. Just as if you invoked Xtags interactively, the script will insert the interpreted text at any insertion point or replace any text selection. The boolean parameters error reporting, document updating and quote conversion correspond to the options in the **Get Text with Xtags...** dialog. 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.

The *source* parameter may be a file alias or a string.

Here is a simple example use of this AppleEvent to import text from a file (we've allowed the lines to wrap):

```
tell application "QuarkXPress™"
    activate
    get text with Xtags from alias "Themis:Text:New Stuff"
    without error reporting with document updating
end tell
```

This example imports the file **New Stuff** in the folder **Text** on disk **Themis** into the current document at the current insertion point, replacing any selected text.

Here is a more complicated script which we've annotated (we've again allowed the lines to wrap):

```
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
```

The **save text with Xtags** event may be used to produce the Xtags text representation for the current text selection (or the whole containing current text box, if requested) or the current picture box, either saving the results in a given file or simply returning it as a string. It has the following general syntax:

save text with Xtags
[**to** ***alias*** *]*
[**with / without containing box** *]*
[**with / without style definitions** *]*
[**with / without separate tags** *]*
[**with / without full list elements** *]*

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(***filespec***)** 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.

The boolean parameter **containing box** requests output of the box-creation tag along with the contents, when in contents mode with a text box selected.

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 (described earlier in this section).

4

Overview of the Xtags Language

General Information

XPress Tags are formatting codes that you place within text files before importing them into QuarkXPress. 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., **<Blz10cK>** is equivalent to **<B><I><z10><cK>** (specifies bold italic ten-point black type). These codes are ***case-sensitive*** in general (though their parameters generally are not). For example, **p** and **P** can mean entirely different things.

Character attribute tags—such as point size and font specifications—take effect immediately, and last until explicitly or implicitly changed. Paragraph attribute 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 attributes applied to a single paragraph, the last attribute applied "wins." E.g., the tag sequence

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

will be centered, not left-justified, since the **<*C>** (centering) tag occurs after the **<*L>** (left-justifying) tag in the same paragraph.

Common ASCII control characters are mapped directly into their QuarkXPress equivalents:

ASCII CODE	CORRESPONDING CHARACTER
7	Shift-return (hard return)
9	Tab
11	Vertical tab (new column)
13	Carriage return (new paragraph)
15	Punctuation space
16	Flexible space
29	Discretionary return

ASCII CODE	CORRESPONDING CHARACTER
30	Temporary margin
31	Soft hyphen

Tag Parameters

Some tags, such as the font size tag shown above, require a parameter. If the parameter is a text string, it must be enclosed in double quotation marks (either straight double quotes or printer's double quotes): for example, **<f"Times">** or **<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: **<f“fontname with \” quote and \\ backslash within it”>**.

When a tag requires a numeric parameter, the number is assumed to be in points by default (as in XPress Tags). 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.

For tags that take only a single numeric parameter—and hence don't normally require parentheses—you must use a special Xtags format to specify the parameter in units other than points: place the parameter in parentheses. For example, **<z(1p)>** sets the font size to 1 pica, i.e., to 12 points, the same as **<z12>**.

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 preceded by a period and then an arithmetic operator (**+**, **–**, ***** or **/**), the value is interpreted as specifying a change with respect to the corresponding current value, with default units corresponding to the kind of value. For example, the tag **<z(.+2)>** tells Xtags to increase the current font size by two points (default units), the tag **<z(.-3p)>** tells Xtags to decrease the current text size by 3 picas (explicit units), and the tag **<s(.–20)>** says to decrease the current shading by 20% (default units, but of a different sort).

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 arguments following ***** and **/** must be unitless numbers.

Xtags supports a "sub-list" parameter specification scheme to simplify the specification of all complex parameter cases. This construct allows you to specify another list inside a list, with the inner list being treated syntactically as a single parameter. Here is an example:

<&tbu2((0, TL, 2), 0, 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 specified a sub-list (highlighted in the tag above) as the first parameter.

We use this scheme to expand the parameter options for some tags in the Xtags 3.x/4.x release 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 more and more baroque little "languages" for parameters that really involve more than one specification (e.g., what used to be **0TL2** now becomes **(0, TL, 2)**, which may not seem like much until you realize that we now have a five-element parameter sub-list for the box height).

Using Default Settings

You can use a dollar sign (**$**) code in place of most XPress Tag parameters, standing for the corresponding 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 corresponding value in the **Normal** style sheet.

Omitted Parameters

Unlike XPress Tags, Xtags allows you to omit parameters from those tags taking multiple parameters. For most tags, Xtags will default any omitted parameters to the value given by default in the corresponding QuarkXPress dialog. For example, if you omit the color parameter in the rule above tag (***ra**), then it will default to **Black**, the same default value used in the **Style→Rules...** QuarkXPress dialog.

Just like list elements, sub-list elements may be omitted to use their default value. For example, the three-element sub-list **(,tl,)** or just **(,tl)** 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, **<b>** is equivalent to **<b0>** (set the baseline shift to zero).

Spacing Within Tags

Xtags is more flexible than XPress Tags about allowing white space—spaces and tabs—within a tag. White space may be placed before and after any ***complete element*** (tag or parameter). For example, the following tags are illegal in XPress Tags but legal in Xtags:

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

while the following tags are ***illegal*** in both XPress Tags and Xtags:

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

because the white space occurs within a number or measurement value (**1 0** and **1 "**) or within a multi-character tag (*** p**).

Both XPress Tags and Xtags allow you to split a long sequence of tags over two or more lines in the text file by placing a colon (but only between tags) at the end of each line 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">
```

An Extended First Example

In this section, we will examine a fairly lengthy example of text formatted with Xtags designed to illustrate the features and power of Xtags. The annotated source text appears on this and the following left-hand (even) page, and the resulting formatted text appears on the right-hand page opposite. Tags are highlighted with boldface text within the source text.

@simple=[S]<f"Helvetica"><z10><P><*p(0,0,0,11,0,12pt)><*J><*kt(2,2)>
This is a style sheet definition, named "simple." 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.

@ruled=[S"simple"]<*ra(2pt,,,50,,,20%)><*rb(2pt,,,50,,,10%)><*p(,,,,6pt)>
This is another style sheet which adds 50% black rules above and below the paragraph to the simple style sheet and increases the space before to 6 points.

@pix=[S"simple"]<*p(,,,,0.75")><*t(2.7",2)><*rb(2pt,,,,,,10%)>
This is a style sheet designed for an ad containing a picture. It has a right tab stop at 2.7" and a black rule below it. It is also based on the simple style sheet.

@sepline=[S"simple"]<B><H><z12><cW><*p(0,0,0,13,12pt,12pt)><*C><*kn1><*ra(14,,,,,,-0.035")>
Based on simple, this style sheet creates reversed separator lines using a 14 point black rule and white text (set in 12 point, bold small caps).

@sepline:apartments for rent *The first separator line*
Now we apply the simple style to the first apartment rental ad. It will remain in effect for the subsequent 2 ads as well.
@simple: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.

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.

Now we have a paragraph where we apply the ruled style sheet (for an ad that is a bit more expensive than the plain ones).
@ruled: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.

Now we go back to the simple style sheet.
@simple: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.

@sepline:houses for rent *The next separator line.*
@simple:Large 3-bedroom townhouse near the University, perfect for a congenial group of students. Kitchen includes all appliances, and there is a deep freeze in the basement. House is located in a sober, non-smoking, vegetarian co-op complex. $650/all utilities plus cable included. Call Rainbow at 555-3422 for further information or to arrange for an interview.

Rambling 5-bedroom house for rent in Woodland Farms area. House has 2 bathrooms and 3+ acres of cultivatable garden area (organic vegetables raised there for the past 7 years). Property is zoned for horses and small livestock. $1250/utilities extra. Call Mitch at 344-2311.

@sepline:houses for sale *The third separator line.*
@simple: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.

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.

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.

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.

HOUSES FOR RENT

Large 3-bedroom townhouse near the University, perfect for a congenial group of students. Kitchen includes all appliances, and there is a deep freeze in the basement. House is located in a sober, non-smoking, vegetarian co-op complex. $650/all utilities plus cable included. Call Rainbow at 555-3422 for further information or to arrange for an interview.

Rambling 5-bedroom house for rent in Woodland Farms area. House has 2 bathrooms and 3+ acres of cultivatable garden area (organic vegetables raised there for the past 7 years). Property is zoned for horses and small livestock. $1250/utilities extra. Call Mitch at 344-2311.

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 on Willbury Hill. 16 room colonial mansion built in 1875, fully updated to the 90's. Glorious gourmet kitchen, 5 bedrooms, 3½ baths. Includes a finished basement with a guest room and a state-of-the-art game and media room. *This property will not last at this price.* $525,000. 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.

This paragraph has the pix style sheet applied. It begins with a 1.25" x 0.75" imported picture—from the file Cons.TIF— scaled to fit the picture box (maintaining the original aspect ratio). The picture is followed by a tab, a bold headline at the right margin, and then the ad text. Within the ad, local character formatting tags create the fraction and the italic phrase.

@pix:<&pb(1.25",.75",,.5pt,,,,,,,A,,,,,,,"Cons.TIF")><\t><B>A Treasure!<B><\n>Magnificent property available on Willbury Hill. 16 room colonial mansion built in 1875, fully updated to the 90's. Glorious gourmet kitchen, 5 bedrooms, 3<z7><b3>1<b0z$f"Symbol">§<f$z7>2<z$> baths. Includes a finished basement with a guest room and a state-of-the-art game and media room. <I>This property will not last at this price.<I> $525,000. Call Janice Snowden to arrange a private showing of this exceptional home, 555-0985, X321.

@simple: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.

About the Tag Descriptions

The sections that follow each describe several related tags. Each tag description includes examples of how it might be used. Since we believe that the best way to understand Xtags constructs is in context, 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 and the final, formatted text produced by importing it into QuarkXPress with Xtags. The source text sometimes includes explicit carriage return and tab characters (denoted by ¶ and →, respectively) to avoid ambiguities. We have highlighted tags in the source text to make them easier to spot. Thus, the text:

@$: Here is an example of the **<B>**f**<B>** tag: **<f"Zapf Dingbats">**u**<f$>** is my favorite bullet character.

would yield the formatted text:

FORMATTED TEXT

Here is an example of the **f** tag: ◆ is my favorite bullet character.

Note that the first paragraph of each example usually begins with the **@$:** tag (see the section on style sheets). This tag applies the **Normal** style sheet to the text, and thus serves to ensure a consistent initial state for each example. In the examples, the **Normal** style sheet produces text that is formatted like the regular text paragraphs within this manual.

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 picture box, or a sample group as you desire, select it, choose **Edit→Copy Xtags Text**,

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.

Similarly, the **Paste Xtags Text** facility may be used to quickly preview the results of some tagged text you are creating. Select the 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.

5

Setting Character Attributes

Character Style Tags

Character style tags are used to specify character attributes of text. These tags generally consist of a single character which toggles the current state of the various character style or "face" attributes: **<P> <B> <I> <O> <S> <U> <W> </> <K> <H> <+> <–> <V> <$>**.

The tag code characters have the following meanings:

P	plain text	**/**	strike-through
B	bold	**K**	all capitals
I	italic	**H**	small capitals
O	outline	**+**	superscript
S	shadow	**–**	subscript
U	underline	**V**	superior
W	word underline	**$**	use current style sheet's character style

The **<P>** tag operates somewhat differently than the others in that it sets the character style to plain text rather than toggling any settings.

We'll now turn to some examples of these character style tags in context (which will be followed by the formatted text).

@$: 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.

@Ital: 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, **<B>**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 TEXT

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.*

Setting vs. Toggling Character Attributes

(Available under Xtags only)

Generally, character style tags toggle character format 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 formats, 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 style attributes.

Character Attribute Tags and Font Selection

Applying the **I** and/or **B** tags to a given font automatically selects any linked fonts in the font family, *but only at output time*. 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.

Character Size and Font Tags

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

<z*size*>
<f"*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. XPress Tags allows you to specify the minimum unique abbreviation for the given font name (though this is not a good idea, as the minimum unique abbreviation may change from session to session, depending on the fonts in use). Note that font name abbreviation is not supported under Xtags, where you must spell out the font name in full.

@$:The font name and size tags are also very easy to use. The **<B>**f**<B>** tag changes the font:

<f"Helvetica">This paragraph is in Helvetica. (We had to spell out "Helvetica" completely, unlike XPress Tags). We can also change the type size: **<z8>**from small to **<z24>**very large **<z$>**with the **<B>**z**<B>** 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 **<B>**$**<B>** parameter, like so, **<f$>**which restores the font to that specified by the current paragraph's current style sheet.

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 TEXT

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, unlike XPress Tags). 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 current 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: H_2O.

Character Color and Shade Tags

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

<c"*color name*">
<c*C*>
<s*shade*>

The **c** tag sets the current text color to the specified document color name. Its second form takes a single letter *C* as a quick abbreviation for common colors. It must be one of: **K** (black), **C** (cyan), **M** (magenta), **Y** (yellow), and **W** (white).

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

@$: The **<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 black 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 do that with tags later.

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.**<Pzs>**

FORMATTED TEXT

The 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 . We'll put a black 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 do that with tags later.

The shade tag (s) controls the percentage shade of the text color. For example:
Here is some big 50% Black text.

Character Scaling, Kerning, and Tracking Tags

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

<h*scale*> *Horizontal scaling*
<y*scale*> *Vertical scaling*

Note that these two tags above are mutually exclusive (you can only scale type in one direction).

<k*kern*>
<t*track*>

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* $^{1}/_{200}^{ths}$ of an em (minimum: -500; maximum: 500). A negative value for *kern* brings letters closer together (kerns in), and a positive value kerns out (pushes letters farther apart).

The **t** tag sets the tracking to *track* $^{1}/_{200}^{ths}$ of an em (minima and maxima are the same as for kerning). A negative value for *track* sets tighter tracking, and a positive value sets looser tracking.

@$: 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.

@$: Here's a kern between two otherwise poorly-spaced letters (note how the kerning is applied only to the first character of the pair to be kerned):

<B><z24><k-10>W**<k0>**e**<z$>** the people.**<P>** You can also scale text, as **<h175>**we do **<y220>**here**<y$>**.

FORMATTED TEXT

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 two otherwise poorly-spaced letters (note how the kerning is applied only to the first character of the pair to be kerned):

We the people. You can also scale text, as we do here.

Character Baseline Shift Tags

There are two tags that control character baseline shift:

<b*shift*> *Absolute baseline shift*
<br*shift*> *Relative baseline shift* *(Available under Xtags only)*

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 downwards. (The maximum shift is 300% in each direction.)

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

@$: Here are some examples. The words in the **<b-2>**middle of this**<b$>** 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:

E = mc**<z(.-3)b2.5>**2**<zb>**

Note that we used a relative value for the font size tag's parameter; it said to decrease the current size by 3 points.

Finally, these tags will shift the baseline up by 25% of the font size, which is **<z12><br25>**3 points**<zbr>**, in this case.

FORMATTED TEXT

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:

$E = mc^2$

Note that we used a relative value for the font size tag's parameter; it said to decrease the current size by 3 points.

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

Insert Special Character Tags

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

<*special*>

Special can take on many values:

<\n>	Line break (“soft return”)
<\d>	Discretionary return
<\–>	Hyphen
<\m>	Em dash†
<\i>	Indent here
<\t>	Right indent tab (***not*** a regular tab—see below)
<\s>	Standard space†
<\f>	Figure (en, half-em) space†
<\p>	Punctuation space†
<\q>	Quarter-em (flexible) space†
<\h>	Discretionary hyphen†
<\2>	Previous text box page number
<\3>	Current text box page number
<\4>	Next text box page number
<\c>	New column
<\b>	New box
<\@>	Literal at sign (@)
<\<>	Literal left angle bracket (**<**)
<\>>	Literal right angle bracket (**>**)
<\\>	Literal backslash (****)
<\#*num*>	The specified QuarkXPress character, where *num* is a decimal value. The most common cases are
<\#13>	paragraph end, and
<\#9>	tab, which don’t have more mnemonic special tags. Other common cases include the decimal value for various extended ASCII characters.†

Items 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.

Note that the **<\t>** tag does not insert a regular tab character; use **<\#9>** to insert a normal tab character.

Select Character Set Tag

The **e** tag selects the character set in use:

<e*n*>

n is a code letter indicating the desired character set: **e0** selects the Macintosh character set, **e1** selects the Windows DTP character set, and **e2** selects the ISO Latin 1 character set. The default is **0** on the Macintosh and **1** under Windows.

The next carriage return following an **e** tag will be ignored on input. Output from Xtags and XPress Tags usually begins with the line:

<e*n*><v*n.m*>¶

XPress Tags Version Tag

<v*n.m*>

The **v** tag specifies the desired version of XPress Tags under which to interpret subsequent tags, where *n* and *m* denote the major and minor version numbers. The default as of this writing is **<v2.0>**.

The next carriage return following an **v** tag will be ignored on input. (See note immediately above about output.)

6

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

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

These tags set the paragraph alignment to left, center, right, or justified, respectively.

@$:<*L>This paragraph is left-aligned,¶
<*C>this paragraph is centered,¶
<*R>this paragraph is right-aligned,¶

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

<*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 TEXT

This paragraph is left-aligned,

this paragraph is centered,

this paragraph is right-aligned,

and 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 syntax:

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

The parameters to ***p** are described individually below:

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).

space before — 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 — 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 (currently ignored by Xtags).

@$: 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 also has 1/4" margins, 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**<fz> 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(1.5p, -1.5p, 0, 11)><f"ZapfDingbats">v**<f$>**→ This paragraph creates a hanging indent by setting the first line indent to a negative value.¶

FORMATTED TEXT

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

Omitted settings don't change, so this paragraph also has 1/4" margins, 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 tab stops—up to 20 under QuarkXPress 3.3, and with no practical limit under QuarkXPress 4.x—using the following syntax:

<*t(*position$_1$*, *alignment$_1$*, "*fill$_1$*", *position$_2$*, *alignment$_2$*, "*fill$_2$*", ...)>

<*t0>	*turns off all tabs (available under Xtags only)*
<*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 the following codes: **0** means left align tab, **1** means centered tab, **2** means right align tab, **4** means decimal-aligned tab, **5** means comma-aligned tab, and "*x*" (where *x* is any character) means align on the specified character (e.g. **"/"** means align on slashes). Note that the quotation marks must be included. (**3** is a deprecated code for decimal-aligned tabs.)

Decimal-aligned tabs and comma-aligned tabs are defined as appropriate for the version of QuarkXPress in use, taking internationalization into account. Thus, to create a tab aligned on a comma character when comma-aligned tabs are defined in some other way, use **","**.

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

Fill characters have the form **"1***xx***"** or **"2***xy***"** (include the quotation marks), where the letters indicate the desired fill character(s). The **1** form specifies a single fill character (repeated twice), and the **2** form specifies two alternating fill characters (the second character will be ignored under pre-3.3 versions of QuarkXPress). 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 "**—one followed by two space characters—to signify no fill (which is also the default if this parameter is omitted).

Here is an example use of the ***t** tag:

@$:<*t(.1", 0, , 1.75", 1, , 3.75", 2, "2. ", 4.4", 4)><*p(,,,,,6)>¶

→(left 0.1")→(centered 1.75")→(right, 3.75", ". " leader)→(dec. 4.4")¶

<*p(,,,,,0)>→45-102-P→Aquastar Pro→40%→$8.75¶

→48-103-H→Aquaseal Deluxe→44%→$128.99¶

→45-202-Q→Aquamaster Underwater→48%→$.75¶

FORMATTED TEXT

(left 0.1”)	(centered 1.75”)(right, 3.75”, “. ” leader)	(dec. 4.4”)
45-102-P	Aquastar Pro 40%	$8.75
48-103-H	Aquaseal Deluxe 44%	$128.99
45-202-Q	Aquamaster Underwater 48%	$.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.

Paragraph Rules Tags

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

<*ra(*thickness*, *style*, "*color*", *shade*, *from left*, *from right*, *offset*)>
<*rb(*thickness*, *style*, "*color*", *shade*, *from left*, *from right*, *offset*)>

<*ra0> *Turn off rules*
<*rb0>

These tags have the same parameters, which are described below:

thickness Vertical thickness of the rule (minimum 0 pt; maximum is the height of the current text box; default 1 pt).

style A numeric code specifying the line style: **0** for solid, **1** for short dashes, **2** for long dashes, **3** for dot-dashes, **4** for dotted, **128** for double line (medium, medium), **129** for double line (thin, thick), **130** for double line (thick, thin), **131** for triple line (thin, thick, thin), **132** for triple line (thick, thin, thick), **133** for triple line (thin, thin, thin). (Default style is **0** (solid).)

color Rule color (default Black).

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

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%).

The special tags **<*ra0>** and **<*rb0>** turn off the corresponding rule altogether.

The following are examples of the paragraph rule tags:

@$: 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.¶

This paragraph has a one-point rule above it, running the width of the column, offset 10 points up.**<*ra(1, , , , , , 10pt)>¶**

<I>Notice:**<I>** This paragraph has thicker 60% grey rules above and below its central area.**<*ra(3, , , 60%, .5", .5", 10pt)*rb(3, , , 60%, .5", .5",)>¶**

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.**<*ra0><*rb(1, 4, , , T0, 0)>¶**

<*ra(16, , , , , , -4.5pt)><*rb0><BcW>This bold text is reversed (white), in front of a black rule.**<c>¶**

FORMATTED TEXT

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.

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.

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:

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

This tag gives the current paragraph a drop cap consisting of the next *chars* characters (minimum 1; maximum 8; default 1), sized to extend for the specified number of *lines* of text (minimum 2; maximum 8; default 1 (the default is, effectively, no drop caps)).

@$: The **<B>***d**<B>** tag creates drop caps. Here is a simple example of a single letter extending over three text lines:

<*d(1,3)>Rarely did Monica miss something so obvious, but when Gerrard slipped the extra cryoprotractor into the pocket of his overalls, she was already lost in the significance of the green blips on her scanner, which seemed to both grow and shrink simultaneously, a feat which had hitherto been unachievable by the enemy.

@$: Drop caps can also consist of more than one character:

<*d(3,2)><BI>For**<P>**nearly two centuries—since Adam Smith published his **<I>**Wealth of Nations**<I>** in 1776—economists have been advertising themselves to the world as the most rigorous and successful of all the social scientists.

FORMATTED TEXT

The *d tag creates drop caps. Here is a simple example of a single letter extending over three text lines:

Rarely did Monica miss something so obvious, but when Gerrard slipped the extra cryoprotractor into the pocket of his overalls, she was already lost in the significance of the green blips on her scanner, which seemed

to both grow and shrink simultaneously, a feat which had hitherto been unachievable by the enemy.

Drop caps can also consist of more than one character:

For nearly two centuries—since Adam Smith published his *Wealth of Nations* in 1776—economists have been advertising themselves to the world as the most rigorous and successful of all the social scientists.

Paragraph "Keep" Tags

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

<*kn*onoff*>
<*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 at the bottom or top of a text column—eminent modern typographers disagree on the exact definition of these terms). The first form, **<*kt(A)>**, requests that all lines in a paragraph be kept together (i.e., appear in a single column).

The second form specifies how QuarkXPress 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.

Here are some examples:

@$:<*p(,12,,,0,0)><*L>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. (We've changed our normal example layout to illustrate these tags more clearly.)**<*kn1>**¶

<*kn0>Putting that tag in the previous paragraph will force text from the first paragraph down into the second text box upon import.¶

<*kt(5,4)>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. For example, we've specified in this paragraph that at least 5 lines must be kept together at the beginning and at least 4 lines must be kept together at the end if this paragraph breaks across a text boundary. We've set it up to break in both places, so you can see the effect of this tag.¶

<*kt(A)>This paragraph's lines must all remain together within one column.¶

FORMATTED TEXT

The ***k** tags specify paragraph keep options. ***kn1** says to keep the current paragraph (or at least part of it) with the next one.

(We've changed our normal example layout to illustrate these tags more clearly.)

Putting that tag in the previous paragraph will force text from the first paragraph down into the second text box upon import.

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. For example, we've specified in this paragraph that at least 5 lines must be kept together at the beginning and at least 4 lines must be kept together at the end if this paragraph breaks across a text boundary. We've set it up to break in both places, so you can see the effect of this tag.

This paragraph's lines must all remain together within one column.

7

Defining and Applying Style Sheets

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

Define Style Sheet Tags

The following tag forms are 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.

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 **No Style**, though all its settings are initialized from the **Normal** style (which is guaranteed to be present in every 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.

Any relative parameter values in **<***tags***>** are computed relative to the based-on style (or to the settings copied from **Normal** if there is no explicit based-on style).

Spaces are significant in *name*. For example, **@ spaces galore =<...>** does *not* define the same style as **@spacesgalore=<...>**. Leading and trailing 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 QuarkXPress document into which you're importing text, Xtags ignores the attempt.

This last point has an important ramification for debugging. 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. Otherwise, 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.

It is impossible to define or redefine the **Normal** style from within an XPress Tags file, since it can not be deleted from any document.

For examples, see the following major subsection.

Defining Character Style Sheets

Character style sheets are defined via very similar 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.)

Examples are included in the next subsection of this manual.

Apply Style Sheet Tags

PARAGRAPH	CHARACTER	
@*name*:	**<@*name*>**	*Apply specified style sheet*
@$:	**<@$>**	*Apply the Normal style sheet*
	<@$p>	*Apply character style of current paragraph style*
@:	**<@>**	*Apply "No Style"*

A paragraph style sheet is applied by placing its name prefixed by an "at sign" (@) and followed by a colon, usually at the beginning of a paragraph. Application involves making all the paragraph and character 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 (though the character attributes take effect at each style sheet application, giving you an easy way to get the input-only effect of "character styles" under QuarkXPress 3.3).

Character style sheets are similarly applied via the **<@*name*>** tag.

The forms **@$:** and **<@$>** are shorthand ways of applying the **Normal** 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** 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**, and applied normally.

Note that no extraneous spaces should be used in or around the stylesheet 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 (assuming **astyle** is a valid style name), while

@a style: We're making two big mistakes.

has two likely problems: there is a space in the style name itself (which will cause a new style with that name to be created, if it doesn't already exist), and there's some space after the colon (which means that there will be some undesired space at the beginning of the paragraph).

We'll now turn to some contextual examples.

@Body=[S"","Body","Normal"]<*J *h"Few" *kt(2,2) *p(.75",0,0,11,0,12)>

@Indent=[S"Body","Indent","Normal"]<*p(.+.35",,.+.35",10) >

@Tip=[S"Indent","Body"]<f"Helvetica" I z9 *ra(3,,,50,,,8) *rb(3,,,50,,,4) <p(,,,10)>

@SName=<f"Helvetica" Bz9>
@TipSName=[S"","","","SName"]<z8>

@Body:Here is the "normal" style sheet in this document, which we've called **<@SName>**Body**<@$>**. It uses very common settings, including the **<@SName>** Normal**<@$>** character style sheet.

@Indent: This paragraph uses another style sheet, **<@SName>**Indent**<@$>**, based on **<@SName>**Body**<@$>**, that brings in both margins a bit. Note that we use the **<@SName>**SName**<@$>** character style sheet to format style sheet names in this example.

@Tip:This style sheet named **<@TipSName>**Tip**<@$p>** is based on **<@TipSName>**Indent**<@$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 **<@TipSName>**Body**<@$p>** as its next style setting. We use the character style sheet **<@TipSName>**TipSName**<@$p>** to format style sheet names in this paragraph.Note that we must turn off this character style sheet by reverting to the **<B>**paragraph**<B>** style sheet's character style.

@Body: 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.

FORMATTED TEXT

Here is the "normal" style sheet in this document, which we've called **Body**. It uses very common settings, including the **Normal** character style sheet.

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

This style sheet named **Tip** *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* **Body** *as its next style setting. We use the character style sheet* **TipSName** *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.

8

Creating Text and Picture Boxes

In QuarkXPress, text and picture boxes are of two types. ***Anchored boxes*** are created by pasting a text or picture box into a text flow. The box then becomes another character within the text story, and it flows along with the accompanying text when margins or pagination changes. By contrast, ***unanchored*** boxes are independent entities which remain in a fixed position on the page unless you explicitly move them.

Xtags can create text and picture boxes of either type. We will consider text boxes first. (All of these tags are available under Xtags only.)

Text Box Tags

The **&tb ... &te** tag pair creates and fills an in-line anchored text box. **&tb** creates the box, and marks the beginning of text to be imported into the box. marks the end of text being imported into the box:

> **<&tb(*width*, *height*, *text align*, *frame width*, *frame color*, *frame shade*, *frame style*, *bg color*, *bg shade*, *text outset*, *columns*, *gutter*, *text inset*, *baseline offset*, *baseline min*, *vert align*, *interparagraph max*, *box name*)>*... Text to be placed in the box, typically containing other tags* ...<&te>**

When an in-line text box is created, all settings that were in effect beforehand remain in effect for text flowing into the box. In other words, text within a created text box inherits all character and paragraph attributes in effect at the time of the box's creation. However, any changes made within an in-line 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. 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.

The **&tbu2 ... &te** tag pair similarly creates an unanchored text box and fills it with text:

> **<&tbu2(*x*, *y*, *width*, *height*, *boxangle*, *boxskew*, *flags*, *runaround*, *frame width*, *frame color*, *frame shade*, *frame style*, *bg color*, *bg shade*, *text outset*, *columns*, *gutter*, *text inset*, *baseline offset*, *baseline min*, *vert align*, *interparagraph max*, *box name*)>*... Text to be placed in the box* ...<&te>**

The parameter lists for these two text box creation tags have many members in common. We now document the meanings, limits, and default values for all of these parameters, noting ones which apply to only one type of text box. Note that parameters whose descriptions end with an asterisk also have more advanced, sub-list forms which are discussed in a later subsection:

x — Denotes the horizontal page-relative position for the new unanchored text box.*

y — Denotes the vertical page-relative position for the new unanchored text box.

Note that both of these parameters are optional. If *x* and *y* are present, then the box is placed with its top left corner at position *x*, *y* on the current page (though "current page" is actually 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 *x*, *y* are absent, then the box is placed "more or less" where it would have been placed, if it had been anchored 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).

If either 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.

width — Text box width (minimum such that column horizontal width, after taking away text inset, any column gutters, and frame width, is 2 pt; maximum 3456 pt; default 72 pt).*

height — Text box height (minimum such that column vertical height, after taking away text inset and frame width, is 2 pt; maximum 3456 pt; default 72 pt).*

text align — Box alignment with respect to the text baseline for anchored text boxes. 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).

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

boxskew The angle with which the text 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). [Unanchored boxes only]

flags An optional string consisting of one or more code letters having the following meanings: **b** (or **B**) = box print suppression, **k** (or **K**) = send the newly-created box behind all other boxes on the spread (as in "send to bac**K**"), and **l** (or **L**) = lock item (as with the QuarkXpress **Item→Lock** menu path). [Unanchored boxes only]

runaround Either **I** for item runaround or **N** for no runaround (these are case-insensitive, as with most of the single-character keys). The default is item (**I**), which is usually what you'd want. [Unanchored boxes only]

frame width Width of the box's frame (minimum 0 pt (none); maximum 504 pt; default 0 pt).*

frame color Color of the box's frame: either a quoted color name or one of the key characters as described under the **<c>** tag previously; default **K** (**Black**).*

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

frame style Style of the box's frame, specified either as the frame style name in quotes (e.g., "**Single**") (under 4.x only) or as a frame style index, an integer which runs from 1 (default) to the number of frames currently installed. See the **Item→Frame...** dialog for a sequential list of the current styles. Note that this frame style index is dependent on the particular QuarkXPress and document preferences currently in use; i.e., beyond the seven built-in frame styles, the other frame style indices and meanings may change as you add new frame styles with the Frame Editor, delete frames, and edit existing frames.

bg color Background color of the box, or **N** for no color.

bg shade Shade of the background color of the box, as a percentage (minimum 0%; maximum 100%). The default for background color and shading is 100% white.

text outset Text runaround exterior margin on all four sides of the box (minimum –288 pt; maximum 288 pt; default 0 pt).*

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

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

text inset The size of the interior text margin on all four sides (minimum 0 pt; maximum 288 pt; default 0 pt).*

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

baseline min 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 min*, see the anchored text box specifications documentation in the QuarkXPress manual.)

vert align Vertical alignment for text within the box; use **T** (or **t**) for top aligned text (the default), **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 An optional AppleEvent-relative box name.

Examples

Here are some examples of creating anchored text boxes:

@$: Xtags uses two tags to create and fill in-line 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 in-line 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 TEXT

Xtags uses two tags to create and fill in-line 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 in-line 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:

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

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.

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.

Here is an example of creating an unanchored text box.

@$: This tag creates a 2" x 0.75" unanchored text box at the bottom center of the page**<&tbu2(2.75", 7.75", 2.5", .75",,, k,,,,,, K, 100)><*p(0,0,0,,0,0)><*C><B><z11>-<cW>**An unanchored text box!**<&te>**. Note that the new text box is placed behind the other boxes on the page, so only a bit of it peeks out.

FORMATTED TEXT

This tag creates a 2.5” x 0.75” unanchored text box at the bottom center of the page. Note that the new text box is placed behind the other boxes on the page, so only a bit of it peeks out.

Shrink-to-Fit Text Boxes

When a text box’s height is given in the form *n*?*m*, where *n* is the initial creation dimension (in any unit, defaulting to points), and *m* is the margin value (in any unit, defaulting to points, and which may be omitted if it’s zero, in which case the height or width value would have the form *n*?), then the box is initially created at the given size, the containing text processed, and once the **<&te>** tag is encountered (when the box contents are finished), the box is examined to see how much height is actually needed, any shrink-to-fit-margins are added to the top and bottom (along with any frame width and text inset), and the box is resized appropriately.

Shrinking-to-fit will fail if the text box is empty or already overflown, if it’s not visible (it’s in some other text box’s overflow text, in the case of an anchored box), or if there is more than one column in the box. Note that text boxes are always shrunk or left alone; they are ***never*** expanded and so must be created at the maximum desired size. You must also 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.

I’m inside...

@$:<*p(,,,,.15",0)>For example, the 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.

@$:<*p(.75",,,,,0)>Similarly, the tag sequence **<&tbu2(5.8",4.0",1.0", 3"?1pt, 90,,, , , , , ,K,15 , 6, , , 4, , , C)><B><*p(0,0,0)><*C><z8>**I'm inside...**<&te>** will create an unanchored box 1.5” wide and initially 3” high, rotate the box by 90 degrees, put the text in the box, and then shrink it vertically to fit, placing it about halfway down the right margin of the page with a six-point runaround.

FORMATTED TEXT

For example, the tag sequence text in the box 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.

Similarly, the tag sequence will create an unanchored box 1.5” wide and initially 3” high, rotate the box by 90 degrees, put the text in the box, and then shrink it vertically to fit, placing it about halfway down the right margin of the page with a six-point runaround.

An unanchored text box!

One caveat to be aware of with respect to the **&tbu** tag (due to a QuarkXPress limitation/quirk) is the following: if you're going to use back-to-back unanchored box tags with no intervening text, then 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.

Picture Box Tags

The *pb tag creates an anchored picture box in a manner analogous to *tb:

<&pb(*width*, *height*, *text align*, *frame width*, *frame color*, *frame shade*, *frame style*, *bg color*, *bg shade*, *text outset*, *placement*, *scalex*, *scaley*, *offsetx*, *offsety*, *angle*, *skew*, *picture pathname*, *picture type*, *box name*)>

This single tag creates an in-line anchored picture box, and optionally imports the picture file named by picture pathname. (Note that when importing a picture, Xtags obeys the TIFF screen resolution(s) specified in the QuarkXPress application preferences dialog.)

Similarly, the **&pbu2** tag creates and optionally fills an unanchored picture box:

<&pbu2(*x*, *y*, *width*, *height*, *boxangle*, *boxskew*, *flags*, *runaround*, *frame width*, *frame color*, *frame shade*, *frame style*, *bg color*, *bg shade*, *text outset*, *placement*, *scalex*, *scaley*, *offsetx*, *offsety*, *pixangle*, *pixskew*, *picture pathname*, *picture type*, *box name*)>

The parameters have the following meanings, limits, and default values (again, those applying to only one type of picture box are so designated). Note that parameters whose descriptions end with an asterisk also have more advanced, sub-list forms which are discussed in a later subsection::

x — Denotes the horizontal page-relative position for the new unanchored picture box.*

y — Denotes the vertical page-relative position for the new unanchored picture box.

Note that both of these parameters are optional. If *x* and *y* are present, then the box is placed with its top left corner at position *x*, *y* on the current page (though "current page" is actually 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 *x*, *y* are absent, then the box is placed "more or less" where it would have been placed, if it had been

anchored 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).

If either 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.

width — Picture box width (minimum is 0.25 pt; maximum 3456 pt; default 72 pt).*

height — Picture box height (minimum is 0.25 pt; maximum 3456 pt; default 72 pt).*

text align — Box alignment with respect to the text baseline for anchored picture boxes. 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).

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

boxskew — 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). [Unanchored boxes only]

flags — An optional string consisting of one or more code letters having the following meanings: **b** (or **B**) = box print suppression, **k** (or **K**) = send the newly-created box behind all other boxes on the spread (as in "send to bac**K**"), and **l** (or **L**) = lock item (as with the QuarkXpress **Item→Lock** menu path). [Unanchored boxes only]

runaround — Either **I** for item runaround or **N** for no runaround (these are case-insensitive, as with most of the single-character keys). The default is item (**I**), which is usually what you'd want. [Unanchored boxes only]

frame width — Width of the box's frame (minimum 0 pt (none); maximum 504 pt; default 0 pt).*

frame color Color of the box's frame: either a quoted color name or one of the key characters as described under the **<c>** tag previously; default **K** (**Black**).*

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

frame style Style of the box's frame, specified either as the frame style name in quotes (e.g., "**Single**") (under 4.x only) or as a frame style index, an integer which runs from 1 (default) to the number of frames currently installed. See the **Item→Frame...** dialog for a sequential list of the current styles. Note that this frame style index is dependent on the particular QuarkXPress and document preferences currently in use; i.e., beyond the seven built-in frame styles, the other frame style indices and meanings may change as you add new frame styles with the Frame Editor, delete frames, and edit existing frames.

bg color Background color of the box, or **N** for no color.

bg shade Shade of the background color of the box, as a percentage (minimum 0%; maximum 100%). The default for background color and shading is 100% white.

text outset Text runaround exterior margin on all four sides of the box (minimum –288 pt; maximum 288 pt; default 0 pt).*

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 *offsetx* and *offsety* as needed to center the picture, fitting of either sort will set *offsetx*, *offsety*, *scalex*, and *scaley* as needed to make the picture fit the containing box, etc.).

scalex Scaling percentage applied to the picture in the horizontal direction (minimum 10%; maximum 1000%; default 100% (meaning original imported size)).

scaley Scaling percentage applied to the picture in the vertical direction (minimum 10%; maximum 1000%; default 100% (meaning original imported size)).

offsetx Horizontal offset of the picture from the left edge of the picture box (minimum is the negative of the native image width; max-

imum is the native image width; default 0 pt (meaning placing the left edge of the picture against the left edge of the box)).

offsety 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; default 0 pt (meaning placing the top edge of the picture against the top edge of the box)).

angle Angle in degrees of the picture within its (unrotated) picture box (minimum -360; maximum 360; default 0 (unrotated)).

skew Picture skew in degrees (minimum -75; maximum 75; default 0 (no skew)).

picture pathname

Quoted file system pathname denoting the picture file to be imported. Xtags supports the full set of picture types supported by the QuarkXPress **Get Picture...** command. The pathname may be either relative to the current document's folder or an absolute pathname. Pathnames are case-insensitive, but quite space-sensitive; e.g., **"Picture"** is the same as **"PICTURE"**, but distinct from **"Pic ture "**. Here are some sample pathnames and their meanings:

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

On the Macintosh, **":Pictures:Leaf.EPS"** is a picture file named **Leaf.EPS** in the folder **Pictures** located in the same folder as the current document. Note the leading colon, which tells Xtags that this pathname is relative to the current document's folder. Under Windows, the corresponding picture file would be specified as **"Pictures\\Leaf.EPS"**.

On the Macintosh **"Disk2:Old_Pix:Flora:Sunfleur.TIF**" is a picture file named **Sunfleur.TIF** located on the disk named **Disk2** in the folder **Flora**, which is itself in the top-level folder named **Old_Pix**. On a Windows system, the corresponding picture file might be **"D:\\Old_Pix\\Flora\\Sunfleur.TIF"**.

If you are unsure about what pathname 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 pathname by using the **Picture Usage** option (QuarkXPress 3.3) or **Usage** option (4.x) on the **Utilities** menu.

picture type A platform-independent key which represents the kind of picture file to be imported: **T** or **t** for TIFF, **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 MacOS 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 An optional AppleEvent-relative box name.

Missing Picture Handling

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

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 **Picture Usage...** (3.3x) / **Usage**... (4.x) 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 MacOS, you must also use the appropriate picture type parameter in the picture box tag (discussed above).

Examples

@$:Here are some examples. Anchored picture boxes work very much like anchored text boxes. The following tag creates a picture box 1.5 inches wide and 1 inch high to hold the picture in the file **<f"Helvetica"z9B>**Cons.TIF **<fzB>** in the same folder as the current document. The picture is sized to fit the picture box, maintaining its original aspect ratio, and has a .5 point solid black frame.

<*C*p(,,,,.85")><&pb(1.5", 1", B, 0.5, , , , , , , A, , , , , , , "Cons.TIF")>

@$: And here is some text following the picture.

@$:<*p(.75",0,0,,,0)>**The following tag creates a picture box 1.4 inches wide and 0.95 inch high to hold the picture in the file **<f"Helvetica"z9B>** Cap.TIF**<fzB> in the same folder as the current document. The picture is sized to fit the picture box, maintaining its original aspect ratio, and has a .5 point solid black frame. The picture box is positioned at the upper left corner of the page with

its runaround set to 6 points.**<&pbu2(.5", .75",1.4", 0.95",,,,,.5pt, , , , , , 6pt, A, , , , , , , "Cap.TIF")>**

FORMATTED TEXT

Here are some examples. Picture boxes work very much like text boxes. The following tag creates a picture box 1.5 inches wide and 1 inch high to hold the picture in the file Cons.TIF in the same folder as the current document. The picture is sized to fit the picture box, maintaining its original aspect ratio, and has a .5 point solid black frame.

And here is some text following the picture.

The following tag creates a picture box 1.4 inches wide and 0.95 inch high to hold the picture in the file Cap.TIF in the same folder as the current document. The picture is sized to fit the picture box, maintaining its original aspect ratio, and has a .5 point solid black frame. The picture box is positioned at the upper left corner of the page with its runaround set to 6 points.

Shrink-to-Fit Picture Boxes

When a picture box's height or width is given in the form *n***?***m*, where *n* is the initial creation dimension (in any unit, defaulting to points), and *m* is the margin value (in any unit, defaulting to points, and which may be omitted if it's zero, in which case the height or width value would have the form *n***?**), then the box is initially created at the given size, the picture is imported, the box is shrunk to fit the picture's appropriate natural dimension (given the x or y scale value and the given margin on both sides of a given dimension (which may be positive or negative)), and then any picture placement is obeyed (the only useful ones in combination with shrink-to-fit are probably **M** and **C** (manual placement & centered)). This shrinking-to-fit is performed for each dimension independently.

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 will fail if the picture box is empty, or if the "shrunken" size would actually expand the box. Both of these will be reported if error reporting is enabled.

@$:<*p(.75",0,0,,0.8",12pt)>For example, this tag **<&pb(1"?-1mm, 2"?-1mm,, 0.5pt, ,,,,,,C,,,,,,,"Ballet.TIF",,)>** 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 TEXT

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.

Conditional Picture Importing

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

@$:<*p(.75",0,0,,.6",0)>**For example, this tag **<&pb(1", 0.75",,,,,,,,,A,,,,,,, "?Mystery .TIF")>** will only create the given anchored picture box if the file **<f"Helvetica"z9B>**Mystery.TIF**<Bzf> exists; otherwise, Xtags will silently proceed without creating anything. That's what happens in this second example **<&pb(1", 2",,,,,,,,,A,,,,,,, "?Not There.TIF")>** where no picture box gets created.

FORMATTED TEXT

For example, this tag  will only create the given anchored picture box if the file Mystery.TIF 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 shrink-to-fit and conditional picture importing features can be used with either anchored or unanchored picture boxes.

Specifying Advanced Box Attributes

Several of the parameters to the box creation tags may be specified as a sub-list, in order to expand the attributes which may be specified, namely:

- the *x* coordinate parameter of any unanchored box creation tag;
- the *width* parameter of any box creation tag;
- the *height* parameter of any box creation tag;
- the *frame width* parameter of any unanchored box creation tag;
- the *frame color* parameter of any box creation tag (Xtags 4.x only);
- the *frame shade* parameter of any box creation tag (Xtags 4.x only);
- the *text outset* parameter of any box creation tag;
- the *text inset* parameter of any text box creation tag.

We will consider the advanced forms of each of the these parameters in the subsections that follow.

Relative Box Placement

We begin with the *x* coordinate parameter for unanchored boxes. Instead of a single coordinate value, this parameter may be specified as a sub-list indicating the box's placement with respect to another box:

(*x-offset*, *relative placement*, *relative box reference*)

where *x-offset* is the offset for relative box placement. *Relative placement* is a code specifying the type of relative placement: **B** or **P** for box placement on the pasteboard, **TL**, **TR, BL**, or **BR** for this box's top left corner placement at the top left, top right, bottom left or bottom right of the referenced box. The actual placement offset by the value of *x-offset*. The *relative box reference* parameter is an integer from 1 to 100, naming the nth-most-recently-created-by-Xtags box as the box to be used as a reference in placing this new box. For example, the sub-list **(3pt,BL,2)** specifies that the top left corner of the current box is to be placed 3 points to the right of the bottom left corner of the second-to-most-recently created box.

Automatic Box Resizing

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

The *width* parameter may be specified via the following sub-list:

(*width*, *sizing specs*, *adjustment*|*margin*, *minimum*)

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

The *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.

Finally, *minimum* is the minimum width permitted by either type of resizing. The maximum width is the initial width of the box, specified by *width*; as before, boxes are never grown as a result of automatic resizing. This value defaults to 0.

For example, the following sublist specifies a picture box whose width will be resized to fit its contents, created with an initial width of 2 inches. 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": **(2.0",S,-2mm,1.5")**.

Similarly, the *height* parameter may be specified as the sub-list:

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

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

The *sizing specs* parameter contains one or more of the following code letters: **S** or **?** for shrink-to-fit-contents sizing, **R** for relative sizing to any relative placement box (see the *x* coordinate specifier), or **L** for forcing the containing paragraph's leading to the final box height (the latter applies to anchored boxes only).

The *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 height permitted by either type of resizing (defaults to 0). 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.

Finally, the *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 spec* includes the **L** code, and it defaults to 0.

For example, the following sub-list 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: **(3.0",SL,1mm,1.0",3pt)**.

Box Frame Specifications

For unanchored boxes, the *frame width* parameter may take the form of the following sub-list:

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

where *frame width* is the width of the frame, and *corner type* is **V** for convex, **C** for concave, or **S** for straight. If the *corner diameter* or *corner type* is specified on a picture box creation tag, then a round-rectangle picture box is created with the given *corner diameter* (defaults to 0) and *corner type* (defaults to convex); these two parameters are ignored for text boxes. For example, the frame width specification **(4, .25", c)** for an unanchored picture box tag will create a frame with a width of 4 points having concave corners with a diameter of one quarter-inch.

The *frame color* parameter of a box-creation tag can be a sub-list:

(*frame fg color***,** *frame bg color***)**

giving you control over both the foreground and background colors in a frame. If you specify only a single color, it controls just the frame's foreground color, as before.

The *frame shade* parameter in a box-creation tag can be a sub-list:

(*frame fg shade***,** *frame bg shade***)**

to give you control over both the foreground and background shades of a box's frame. If you specify a single shade, it controls just the frame's foreground shade, as before. The *frame bg color* may be specified as **N** to select a **None** color (as in other color cases).

Text Insets and Outsets

The *text inset* (for text boxes) and *text outset* parameters have been enhanced to enable you to specify the four amounts individually. Both of these sub-lists have the following form:

Our nation's capital

(*top*, *left*, *bottom*, *right*)

Any omitted value defaults to 0.

Example

The following an example illustrates the use of many of these enhanced tag parameters:

@$:<*p(0.75",,,,0)>Here is an example of a picture and caption, positioned in the upper left portion of the page. The 1.25"x0.75" picture box is created first. **<&pbu2(.25",.75", (1.25",S,0,0), (.75",S,0,0),,,,,(3.5pt,1pt,C),,50,,,,6pt,,,,,,,, "Cap.TIF")>** The picture is resized to fit to fit the picture both horizontally and vertically. The box has a 4 point, 50% black frame with concave rounded corners of 1 point. We specify a right side text outset of 6 pts and a 1 point text outset on the bottom.

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,10pt),,,0.5pt)><*p(0,0,0,,0,0)><*C><z8l>**Our nation's capital**<&te>**. We use a bottom text outset of 12 points, a right side text outset of 10 points, and a text inset setting of 0.5 points on all four sides.

FORMATTED TEXT

Here is an example of a picture and caption, positioned in the upper left portion of the page. The 1.25”x0.75” picture box is created first. The picture is resized to fit to fit the picture both horizontally and vertically. The box has a 4 point, 50% black frame with concave rounded corners of 1 point. We specify a right side text outset of 6 pts and a 1 point text outset on the bottom.

After the picture box is created, a small text box is created just below it using relative positioning and filled with a caption. We use a bottom text outset of 12 points, a right side text outset of 10 points, and a text inset setting of 0.5 points on all four sides.

Group Unanchored Boxes Tag

The **&g** tag is used to group unanchored Xtags-created boxes of either type. It has this format:

<&g(*n*$_1$, *n*$_2$, ...*n*$_m$)>

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 single box.

For example, the following tag would group the two boxes we created in the preceding subsection:

```
<&g(1, 2)>
```

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

Groups, including recursive (nested) groups—but not a set multiple-selected objects—may be input and output with the normal **Get Text with Xtags** and **Save Text with Xtags** items on the **Edit** menu, in item tool mode.

9

Applying Master Pages

Xtags includes tags to apply a master page to the current document page or spread.

Apply Master Page Tags

(Available under Xtags only)

<&m"*masterpage*">	*apply master to current page/spread*
<&mf"*masterpage*">	*apply master to current page/spread*
<&mp"*masterpage*">	*apply master to current page only*
<&mpf"*masterpage*">	*apply master to current page only*

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, but, basically, 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."

@$: Master page application tags are difficult to demonstrate in a limited space, so we've made a special master page for that purpose which gets applied to this manual page.**<&m"Demo Master">**

FORMATTED TEXT

Master page application tags are difficult to demonstrate in a limited space, so we've made a special master page for that purpose which gets applied to this manual page.

@$: Our next example uses tiny, tiny pages, which we reproduce in full. 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 TEXT

PAGE 1

Our next example uses tiny, tiny pages, which we reproduce in full. The default master page is master page A.

PAGE 2

Once we force our way to page 2, the &m tag will apply master page C to it. Then, we will once again force a new page break.

PAGE 3

The third page has three master page tags on it. The final &m takes precedence over all the others. An &mf tag determines the master page *only* when there are no &m tags

10

Using Translation Tables

The Xtags translation table facility allows you define a translation table 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.

Use Translation Table Tag

(Available under Xtags only)

<&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).

The string parameter *table name* must be the name of a file, which is looked for in the following folders, 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 **System Folder:Preferences:Xtags** under MacOS (i.e., the **Xtags** folder in the **Preferences** folder in the **System Folder**) or in the working directory under Windows;
4. an **Xtags** sub-folder of any path specified in the **emxtagspath** environment variable under Windows.

Translation Table Format

A translation table is an ordinary text (ASCII) file created by any application (be sure to save the text as a raw text file if you enter 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 on the Macintosh or under Windows with automatic character set translation. The character set codes are the same as for the XPress Tags **<e*n*>** character set selection tag: **e0** selects the Macintosh character set, **e1** selects the Windows DTP character set, and **e2** selects the ISO Latin 1 character set.

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:

\r	*carriage-return/return (ASCII 13)*
\t	*tab (ASCII 9)*
\b	*backspace (ASCII 8)*
\n	*newline/linefeed (ASCII 10)*
****	*literal backslash character*

Note that 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 QuarkXPress tabs and paragraph end characters (respectively) 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:

«MDNM»	***chevron marks***
{BP}	***curly braces***
[ep]	***square brackets***
%SIMS%	***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 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 the translation table for our example.

```
; File: "HD80:System Folder:Preferences:Xtags:Example TT"

; Define some special character escapes.
!b          •          bullet
!B          ß          beta
!d          †          dagger
!dd         ‡          double dagger

; Only allow explicit [ep] tags to denote end of paragraph, not returns.
\r          <\#32>     ASCII carriage return becomes a space
[ep]\r      <\#13>     ep tag at end of line becomes  para. end
\\[ep]      [ep]       \[ep] becomes literal version [ep]

; Define some HTML-like tag bracketing pairs for "logical" tagging.
[a]         <B>        author begin
[/a]        <B>        author end
[t]         <I>        title begin
[/t]        <I>        title end

; A source string that turns into nothing.
[empty]
```

Here is the example input file and results (we've highlighted both Xtags and translation constructions in the source text):

<&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 HTML-like tags.**[ep]**

FORMATTED TEXT

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 HTML-like 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 **Thisisatarget string** that will replace the source string **%param1%** in sub-

sequent 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).

As an example, consider the input:

<&tt2e("%1%","First")>This is the %1% attempt.

Importing it will result in the following output:

FORMATTED TEXT

This is the First attempt.

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 translate a return to the literal **{return}**, you'd use something like **&tt2e("\\r", "{return}")**.

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 (naturally).

11

Using Macros

Xtags support a macro facility, based on two tags used to define and execute macros.

Macro Definition and Invocation Tags

The **&!** tag is used to define a macro:

<&!(*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 "**!**"

<!*macroname*(*arg*$_1$, *arg*$_2$, *arg*$_3$, ..., *arg*$_n$)>

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. 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 (of either flavor).

Defining Macros

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:

!*n*	*the entire n^{th} argument, for n=1 through 9*
!(*n*)	*the entire n^{th} argument, for any value of n*
!(*n f*)	*the f^{th} character of the argument n*
!(*n f*:)	*the f^{th} through the last character of the argument n*
!(*n f*:*l*)	*the f^{th} through the l^{th} character of the argument n*

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 *f* or *l* 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 input to QuarkXPress, the macrobody will have to contain a **>** and then a subsequent **<** tag, to leave command mode and later reenter 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,bl,1),0,2\",(1\",s))>@
caption:!1<&te")>
```

when invoked with **<!build("test.tif")>**, will expand to:

```
<&pbu2(0,0,2",1",,,,,,,,,1,,,1,c,,,,,,,"test.tif",,)
&tbu2((0,bl,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).

Examples

The following macro definitions and invocations produce the indicated results. Note the initial and final > and < 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 use 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), which, when used in **<!fm1(32,457)>**, produces

```
<+>3<k-10>2<+k-5>/4<k-10>57<>
```

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:

$^{1}/_{2}$ $^{3}/_{200}$ $4^{3}/_{8}$ $^{32}/_{457}$

This fraction macro assumes that the superscript **VScale** and **HScale** settings in the **Edit→Preferences→Typographic...** 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 you needn't put all of your macro definitions in every input file, if you're using a translation table. 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 character
```

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 1/2 cup of flour to the mixture and then slowly pour in 3/4 cup of scalded milk.

12

Creating Xcatalog Links

Xtags supports three Xcatalog link creation tags, as follows. (To make use of or even any sense of these tags, you must be using Xcatalog 3.x/4.x as well):

&Cs *delimits the start of a text selection Xcatalog link*
&Ce(*dd*, *flags*, *key*, *keytype*, *field*, *subfield*, *picpos*, *pricestyle*)
delimits the end of a text selection Xcatalog link and supplies the link information
&Cb(*relative box ref*, *dd*, *flags*, *key*, *keytype*, *field*, *subfield*, *picpos*, *pricestyle*)
creates a box-based Xcatalog link for a relatively-referenced box

&Cb creates a box-based Xcatalog link for the box referenced by *relative box* (**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 Xcatalog 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).

keytype — 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.

picpos — Picture positioning code: **S** for “As-Is”, **M** for “Upper Left” (manual placement, to parallel the **&pb** and **&pbu2** tags picture

placement), **C** for “Centered”, **F** for “Fit to box”, and **A** for “Fit to box maintaining aspect ratio” (corresponding to the five choices in the Xcatalog linking palette pop-up menu for picture placement) (default is **M**).

pricestyle 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), field “picture path”, with “as is” positioning, you’d use a tag sequence like:

<&pbu2(...)&Cb(1, Catalog, , *, g, “picture path”, , s)>

13

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.

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 (this is how the XPress Tags filter works).

Error Alerts

Xtags encountered 1 error during import (the text «Xtags error: description» is left at the point of error).

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*»**.

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.

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.

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). If this isn't the problem, then perhaps the suitcase containing the font isn't currently loaded by Suitcase or Master Juggler, if you use either of those utilities.

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. (Again, if you use Suitcase or Master Juggler, perhaps the font's suitcase isn't currently loaded.)

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.

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.

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).

Too many tabs.
More than 20 tabs are being defined in a ***t** tag.

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.

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.

Not enough memory for translations.
Xtags is attempting to expand its translation table and can't allocate enough memory to do so. Try running QuarkXPress with a larger memory allocation.

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.

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"<...>
```

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 next style.
The next-style given in a style definition is not a known style in the current document (it may be misspelled).

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 "**1***xx*" or "**2***xy*".

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 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).

Nested macro invocation unsupported.
A nested macro invocation was found and is not supported by Xtags, currently (e.g., **<!m1("!m2(a)", b)>**).

No such macro.
A macro was invoked that hasn't been defined. Check your spelling.

Macro body too long.
A macro's definition body won't fit in the prescribed limit (4096 characters, currently).

Not enough memory for macro definition.
Xtags is attempting to add a macro definition and can't allocate enough memory to do so. Try running QuarkXPress with a larger memory allocation.

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).

Macro expansion too long.
A macro being expanded at invocation time would be too long for the available space (4096 characters, currently).

Can't add style to document.
For some reason, defining a new style (**@...=...**) failed. Most likely, the maximum number of styles per document (128) has been exceeded under QuarkXPress 3.3 (there is no such limit under 4.x).

Can't add style: no memory.
Defining a new style (**@...=...**) failed because there's not enough memory to hold the temporary new style. Try increasing QuarkXPress' memory allocation.

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 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 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 find picture file.
Xtags can't find the picture file as named in a **&pb**/**&pbu2** tag.

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.)

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).

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>**).

Text boxes nested too deep.
You've nested text box creation beyond the limit (8).

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.

Unmatched &te.
A **&te** tag has appeared without a preceding matching **&tb**.

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.

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).

No such previous box.
A relative box reference is within a valid range, but refers to a box we haven't created (yet).

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 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 is empty.
A shrink-to-fit text box is empty, and it can't be shrunk to zero height.

Can't fit: too much text.
A shrink-to-fit text box would have to expand, not shrink, to accommodate its current contents.

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: text box not visible.
A shrink-to-fit text box is in overflowed text and can't be shrunk to fit.

Can't fit: general problems.
A generic error for shrink-to-fit failures (e.g., out of memory, XPress failure, etc.).

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.)

Toolbox or XPress error.
Generic failure (no specifics known).

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.

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.

No current document.
There is no current document, so no import is possible.

No current box.
There is no current box in the current document, so no import is possible.

Current box is not text box.
The current box in the current document isn't a text box, so no import is possible.

Index

Tags and Symbols

X