



**BroadVision<sup>®</sup> QuickSilver<sup>™</sup>**

**File Formats**

File Formats

Copyright © 1998-2002 BroadVision, Inc. All rights reserved.  
585 Broadway, Redwood City, California 94063 U.S.A.  
Printed in the United States of America

This manual and the software described in it are copyrighted.  
Under the copyright laws, this manual or the software may not be copied, in whole or in part, without prior written consent of BroadVision, Inc., or its assignees, except for purposes of internal use by licensed customers of BroadVision, Inc.. This manual and the software described in it are provided under the terms of a license between BroadVision and the recipient, and their use is subject to the terms of that license.

RESTRICTED RIGHTS LEGEND: Terms and Conditions Applicable to Federal Governmental End Users. BroadVision licenses products for ultimate end use by federal government agencies and other federal government customers ("federal government customers") only under the following conditions. Software and technical data rights in these products include only those rights customarily provided to end use customers of Software as defined in the BroadVision, Inc. Software Subscription License Agreement and any exhibit thereto. This customary commercial license in technical data and software is provided in accordance with FAR 12.211 (Technical Data) and 12.212 (Computer Software) and, for Department of Defense purchases, DFAR 252.227-7015 (Technical Data - Commercial Items) and DFAR 227.7202-3 (Rights in Commercial Computer Software or Computer Software Documentation). If a federal government or other public sector customer has a need for rights not conveyed under these terms, it must negotiate with BroadVision to determine if there are acceptable terms for transferring such rights, and a mutually acceptable written agreement specifically conveying such rights must be executed by both parties.

The product described in this manual may be protected by one or more U.S. and International patents. Certain applications of BroadVision One-To-One® software are covered by U.S. patent 5,710,887.

DISCLAIMER: BroadVision, Inc. makes no representations or warranties with respect to the contents or use of this publication. BroadVision shall have no liability for lost profits or other incidental or consequential damages even if advised of the possibility of such damages or for any claim by any third party. Further, BroadVision, Inc. reserves the right to revise this publication and to make changes in its contents at any time, without obligation to notify any person or entity of such revisions or changes.

TITLE: Title and ownership shall at all times remain with BroadVision (and its suppliers if applicable). Use is subject to a separate license agreement with BroadVision.

TRADEMARKS: BroadVision® and BroadVision One-To-One are registered trademarks of BroadVision, Inc., in the United States and the European Community, and are trademarks of BroadVision, Inc., in other countries. The BroadVision logo, Interleaf®, Interleaf 5, Interleaf 6, Interleaf 7, BladeRunner, RDM, WorldView, WorldView Press, and QuickSilver are trademarks of BroadVision, Inc., in the United States and other countries.

The following are trademarks or registered trademarks of their respective companies or organizations in the United States and other countries:

- Acrobat, the Acrobat logo, Distiller, FrameMaker, and PostScript / Adobe Systems Incorporated
  - HP / Hewlett-Packard Co.
  - Java and all Java-based trademarks and logos / Sun Microsystems, Inc.
  - Microsoft, Microsoft Word, MS-DOS, and Internet Explorer / Microsoft Corp.
  - Motif, OSF/Motif, and the OSF/1 operating system / Open Software Foundation, Inc.
  - Netscape, Netscape Navigator, and the Netscape Communications logo / Netscape Communications Corp.
  - RISC System/6000 / International Business Machines Corp.
  - Sun, Sun-4, SunOS4, SunOS5, Solaris, and SunWindows / Sun Microsystems, Inc.
  - UNIX / The Open Group
  - Windows, Windows for Workgroups, Windows NT, Windows 95, Windows 98, and Windows 2000 / Microsoft Corp.
  - X Window System / Massachusetts Institute of Technology
- All other trademarks, service marks, and trade names belong to their respective owners. BroadVision, Inc. disclaims any proprietary interest in the marks and names of others.

.....  
This documentation was prepared using QuickSilver.  
.....

# Contents

## File Formats

---

<b>About this Manual</b> .....	<b>iii</b>
Conventions Used in this Manual .....	iv
Changes in Interleaf 7 .....	v
 <b>1. ASCII Format Basics</b> .....	 <b>1-1</b>
Overview of Interleaf ASCII Format .....	1-2
Declarations, Commands, and Properties .....	1-3
Declarations .....	1-3
Commands .....	1-4
Properties .....	1-4
Default Values .....	1-4
Required Order of Declarations .....	1-5
Syntax for Interleaf ASCII Format .....	1-7
General Conventions .....	1-7
Quoting Conventions .....	1-8
“Loose Matching” .....	1-9
Units of Measurement .....	1-10
Conventions for Specifying Properties .....	1-10
The Interleaf ASCII Loader and ASCII File Type .....	1-11
Files Without Markup .....	1-11
File Type: Icons and Extensions .....	1-12
Character Mapping .....	1-13
ASCII Error Messages .....	1-13
Creating and Editing ASCII Format Documents .....	1-14
Information Discarded by Interleaf ASCII .....	1-14

<b>2. ASCII Format for Text .....</b>	<b>2-1</b>
Declarations .....	2-2
<!OPS, ...> Declaration .....	2-2
<!Page Number Stream, ...> Declaration .....	2-2
<!Document, ...> Declaration .....	2-4
<!Page, ...> Declaration .....	2-7
<!Font Definitions, ...> or <!Fonts, ...> Declaration .....	2-11
<!Color Definitions, ...> or <!Colors, ...> Declaration .....	2-13
<!Pattern Definitions, ...> or <!Patterns, ...> Declaration .....	2-15
<!Revision Tracking, ...> Declaration .....	2-17
<!Autonumber Stream, ...> Declaration .....	2-18
<!Class Defaults, ...> Declaration .....	2-19
<!Class, ...> Declaration — Declaration for Components .....	2-20
<!Master Frame, ...> Declaration .....	2-31
<!Master Diagramming Object,...> Declaration .....	2-38
<!Master Table,...> Declaration .....	2-38
<Master Row... > Declaration .....	2-38
<!Comment, ...> Declaration .....	2-38
<!End Declarations> Declaration .....	2-38
Commands .....	2-39
<Autonum, ...> Command — Autonumbers .....	2-39
<Ref, ...> Command — Reference Tokens .....	2-40
<component, ...> Command .....	2-42
Inline Components .....	2-45
Examples .....	2-48
<Table, ...>, <Row, ...>, and <Cell, ...> Commands .....	2-49
<Fn> Command — Font Change .....	2-49
Text Properties .....	2-50
</F> Command — Previous Font .....	2-52
<FI> Command — Italic .....	2-53
<FB> Command — Bold .....	2-53
<FBI> Command — Bold Italic .....	2-53
<FJ> Command — Force-Justify .....	2-53
<Frame, ...> Command .....	2-54
<HR> Command — Hard Return .....	2-55
<SR> — Soft Return .....	2-55
<!Include, ...> and <!Include Declarations, ...> Commands .....	2-55
<Index, ...> Command .....	2-56
< , ...>, Page Break Command .....	2-57

<Page Header, ...> and <Page Footer, ...> Commands .....	2-58
<SP> Command — Hard Space .....	2-59
<Tab ...> Command .....	2-59
Special Characters: the <#xx> Command .....	2-60
User-Defined Attributes .....	2-61
<Comment ...> Command .....	2-63
<b>3. ASCII Format for Tables .....</b>	<b>3-1</b>
Table Declarations and Commands .....	3-2
<!Master Table, ...> Declaration .....	3-2
<Master Row, ...>Declaration .....	3-7
<Table, ...> Command .....	3-8
<Row, ...> Command .....	3-8
<Cell, ...> Command .....	3-9
Sample Tables .....	3-11
<b>4. ASCII Format for Graphics Objects .....</b>	<b>4-1</b>
General Markup Rules .....	4-2
Object Type and Version .....	4-3
Layering .....	4-4
Locks .....	4-4
Edge and Fill Properties .....	4-6
Graphics Error Messages .....	4-6
ASCII Lisp Method Storage for Graphic Objects .....	4-7
Markup for Specific Objects .....	4-8
Lines .....	4-8
Groups .....	4-9
Polys .....	4-10
Ellipses .....	4-11
Arcs .....	4-12
Splines .....	4-15
Text Strings .....	4-17
Convert-to-Outline Object .....	4-18
Microdocuments .....	4-19
Plotter or Vector-List Object .....	4-21
Encapsulated PostScript Object .....	4-23
OLE Objects .....	4-25
Bézier Objects .....	4-27
Named Graphics Objects .....	4-28
Edit State Objects .....	4-31
Charts and Images .....	4-32

<b>5. ASCII Format for Charts .....</b>	<b>5-1</b>
Record Structure .....	5-2
Units in ASCII Markup for Charts .....	5-2
ASCII Record Markup for Charts .....	5-3
Records for Chart Style Control .....	5-4
Data Records .....	5-9
Records for Variable Style Control .....	5-10
Sample Chart .....	5-13
 <b>6. ASCII Format for Image Objects .....</b>	 <b>6-1</b>
Image Object Markup .....	6-2
Color Image Object Markup .....	6-10
Format Fields and Values .....	6-12
Backing Store Object Types .....	6-16
 <b>7. Binary Format for Image Objects .....</b>	 <b>7-1</b>
Image Object Binary Format .....	7-2
Structure of an Interleaf Image File .....	7-2
Pasting a Raster File into a Document .....	7-4
Color Image Binary Format .....	7-4
Structure of Interleaf Image Files .....	7-5
Compressed Raster Format .....	7-8
 <b>8. The Include Commands .....</b>	 <b>8-1</b>
Using the Include Commands .....	8-2
Using Pathnames .....	8-2
Nesting Include Commands .....	8-2
Using Templates .....	8-2
Assembling a Document .....	8-6
Inserting Frames in an ASCII File .....	8-7

<b>A. Default Values for ASCII Format Documents .....</b>	<b>A-1</b>
Default Values .....	A-2
<b>B. ASCII Format Error Messages .....</b>	<b>B-1</b>
<b>C. Hexadecimal Codes .....</b>	<b>C-1</b>
<b>D. Filter Writing .....</b>	<b>D-1</b>
Autonumbers .....	D-1
Components (<!Class, ...> objects) .....	D-2
Default Values .....	D-2
Diagramming Markup .....	D-3
End Commands .....	D-3
Fill Property .....	D-3
Font Tokens and Font Inheritance .....	D-4
Frames .....	D-5
Informational Markup .....	D-5
Input Markup .....	D-6
Interleaf ASCII Method Storage .....	D-6
Microdocuments .....	D-6
Names .....	D-6
Newlines and Broken Words .....	D-7
Quotation Procedures .....	D-8
<b>Index .....</b>	<b>Index-1</b>
<b>Properties .....</b>	<b>Index-7</b>
<b>Special Cases .....</b>	<b>Index-12</b>

# *About this Manual*

## **Interleaf ASCII Format**

This manual describes **Interleaf ASCII format** for Interleaf 7. Interleaf ASCII format, or **markup**, is a language that describes an Interleaf document.

Interleaf ASCII format and the *File Formats* manual are often used to move documents between Interleaf 7 and other applications. Typically, this involves the use of a filter program to translate between Interleaf's ASCII format and a representation native to some other application. This manual is intended to assist software professionals in the writing of these filter programs.

## **Interleaf Binary Format**

**Interleaf binary**, or **fast**, format, is another Interleaf format for describing a document. Unlike ASCII format, Interleaf binary format is proprietary; programs other than the Interleaf publishing software cannot easily use it. This manual discusses Interleaf binary format only in relation to images before they are pasted in documents.

## **Chapters**

The chapters in this manual describe

- ASCII format basics
- ASCII format for text
- ASCII format for tables
- ASCII format for graphics objects
- ASCII format for charts
- ASCII format for image objects in a document
- binary format for images not yet pasted in a document
- the Include commands

.....



## Appendixes

The appendixes in this manual describe

- the default values for ASCII format
- ASCII format error messages
- hexadecimal codes for markup of special characters
- hints for filter writers

## Indexes

In addition to the main index, there is a *Properties Index* that lists all properties that can be declared in Interleaf ASCII format, and a *Special Cases* index that lists cases that vary in some way from standard format declarations or procedures.

This manual gives detailed information about Interleaf ASCII format, and briefly describes the properties and values that you can define in ASCII format. For detailed descriptions of these properties and values, and for information about Interleaf-specific terms such as “inline,” “component,” and “frame,” refer to the Interleaf 7 online documentation.

---

## Conventions Used in this Manual

In this manual, the conventions for values of properties are:

- *x* (as in *Top Margin = x inches*) indicates numbers that can contain decimal fractions.
- *n* (as in *Fn = font name*) indicates whole numbers only.
- *m* (as in *Fm = font name*) also indicates a whole number, but indicates in addition that this number must be different from the one preceding it.
- *name* (as in *Final Output Device = name*) indicates that you must enter a name. For naming guidelines, see *Quoting Conventions* in this chapter.
- *string* (as in *Page # Prefix = string*) indicates that the user can insert a string or leave the entry blank. *Strings*, unlike *names*, often contain special characters. To determine when it is necessary to enclose a *string* entry in quotation marks, see *Quoting Conventions* in this chapter.

In examples of ASCII markup throughout the manual, default values are shown in boldface type like this:

```
<!Document,
    Header Page = On/Off,
    Double Sided = Yes/No,
    Manual Sheet Feed = Yes/No,
    Print Rev Bars = Yes/No,
    .
    .
    ...>
```

---

## Changes in Interleaf 7

If you save a document in Interleaf 7 ASCII format, the markup differs from previous releases in the following ways:

- The ASCII version number is 8.4. The markup is:

```
<!OPS, Version = 8.4>
```

Documents produced in Interleaf 7 can thus be differentiated from older versions. The ASCII loader issues errors only if it sees a version number with a whole number step, such as 9.0.

- ASCII format for compressed color images. Changes to compressed backing store object markup.
- Color Images. The format is changed so that Interleaf supports depths of 1 bit/pixel (for line-art images), 8 bits/pixel (for grey-scale or 'pseudocolor' contone images) and 24 bits/pixel (for color images).

# *ASCII Format Basics*

# *1*

This chapter covers basic concepts of Interleaf ASCII format by presenting the following:

- overview
- declarations, commands, and properties
- required order of declarations
- syntax for Interleaf ASCII format
- the Interleaf ASCII loader and ASCII file type
- character mapping
- ASCII error messages
- creating and editing ASCII-format documents

---

## Overview of Interleaf ASCII Format

### Interleaf ASCII Format

Interleaf ASCII format, or markup, is a language that describes an Interleaf document. An Interleaf document can contain structuring elements such as components and frames. These can contain text, graphics objects, charts, images, and other elements.

An Interleaf ASCII format file typically contains both markup and ASCII text. The markup specifies formatting information for the contents of an Interleaf document. This chapter presents a summary of Interleaf ASCII markup syntax and procedure.

### ASCII Loader and Dumper

Throughout this manual, reference is made to the Interleaf ASCII loader and dumper. **Loader** refers to the subsystem of Interleaf that reads in, or loads, an ASCII file to the application. **Dumper** refers to the subsystem that writes out an Interleaf binary file in Interleaf ASCII format.

### ASCII Lisp Method Storage for Components

Starting with Release 5.2, components can have Interleaf Lisp statements attached on output. When Lisp is present in output, it is expressed using the property syntax, as in the following example:

```
<component ,
    ...
    Lisp = "actual Lisp code"
    ...
```

Lisp code must be enclosed between quotes. If the Lisp code itself contains either a double quotation mark (") or a backslash (\), either character must be preceded by a backslash. Lines of output are kept short in length by outputting a backslash and a newline. If the Lisp string contains actual newline or tab characters, they will be preserved as written in output. If there are characters with values greater than *hex 7e*, they are output as `\xHH`, where *HH* stands for two hex digits.

For example, executing the following Interleaf Lisp:

```
(tell cmpn mid:put-saved-data :foo "(<>,\\"\\ \" )
```

results in the following markup:

```
<"para",
  Lisp = "(tell *load-object* mid:put-data
    'ileaf::saved-data \ '(:foo \\"(<>,\\"\\\\"\\ \" )
  )" >
```

Lisp can also be attached to graphics objects. For more information, refer to the chapter *ASCII Format for Graphics Objects*.

---

## Declarations, Commands, and Properties

Interleaf ASCII markup uses two kinds of statements: declarations and commands.

### Declarations

**Declarations** define the properties of the entire document and the properties of individual document elements. Declarations must precede all other content. Each declaration is enclosed by the symbols `<!` and `>`. The following is an example of a partial declaration for a document:

```
<!Document,
  Header Page = no,
  Manual Sheet Feed = yes,
  Print Underlines = no,
  Underline at Descender = yes,
  Default Page Stream Name = page>
```

The main function of a declaration is to establish master definitions for the structuring elements of an Interleaf document, such as components or frames. Individual instances of a master can have unique features, but they rely on the master declaration for most of their structuring information. For this reason, markup for a specific instance should not precede its master declaration.

## Commands

**Commands** appear throughout the file to specify format changes and to indicate the start of a structuring element (such as a component or a frame) previously defined by a declaration. Each command is enclosed by the symbols < and >. The following example shows a command for a *para* component:

```
<para,  
    Top Margin =          1 Inches,  
    Bottom Margin =       1 Inches,  
    Left Margin =         0.50 Inches,  
    Right Margin =        0.50 Inches,  
    Allow Page Break Within = no>
```

## Properties

Both declarations and commands specify **properties** and their **values**. For a list of all properties and their corresponding declarations or commands, refer to the *Properties Index*.

## Default Values

If you do not provide a value for a property in an input file, Interleaf 7 supplies a default value. For a list of all default values, refer to the appendix *Default Values for ASCII Format Documents*. For special cases, see Appendix D, *Filter Writing*.

The default property values listed in this appendix may not be the same as the default values that appear when you create a new document with the Interleaf 7. This is because defaulting priorities differ in these two areas. In interactive work, defaults provide a consistent set of properties within a document and across a set of documents. These user visible defaults can be edited and reflect individual preferences, an organizational standard, or the conventions of a country.

The ASCII dumper and loader use defaults to reduce the volume of markup. Space can be saved by generating markup only when a property has a value different from the default. For example, most components do not have `Begin New Page = yes`, so when a master component is output, the `Begin New Page` property is only mentioned when it is `yes`.

Because the ASCII Loader/Dumper uses defaults this way, the defaults are kept the same from one release to the next, unless the change is clearly linked to the version number.

---

**Note**

If you intend to write a program to generate ASCII markup, refer to the ASCII default values for properties in Appendix A, *Default Values for ASCII Format Documents*, since they may differ from the ones you see in Interleaf 7.

---

## Required Order of Declarations

The order in which Interleaf ASCII declarations are assembled for markup is important, since some declarations use information provided in earlier declarations. For example, a component containing text needs information about fonts. Therefore, fonts must be defined before they can be referred to by any text components. This means the *<!Font Definitions, ...>* declaration must precede any *<!Class, ...>* declarations for components.

The required order of declarations in an ASCII file is as follows:

- *<!OPS, ...>* is mandatory when there are no other declarations in the document, and is strongly recommended when there are other declarations. If used, it must precede any other declaration.
- *<!Page Number Stream, ...>* must precede any other declaration that makes reference to that named stream. This includes *<!Document, ...>*.
- *<!Document, ...>* must follow *<!Page Number Stream, ...>*.
- *<!Font Definitions, ...>*. The legality of fonts that you declare for your document depends on the printer type specified in the *<!Document, ...>* declaration. The declarations that follow depend on the font definitions.
- *<!Page, ...>* declarations, which are also used in microdocuments.
- *<!Autonumber Stream, ...>* declarations must appear before the *<!Class Defaults, ...>* and *<!Class, ...>* declarations.

- *<!Color Definitions, ...>* and *<!Pattern Definitions, ...>* may be omitted when only the default color and pattern choices supplied by Interleaf are used. Otherwise, they must follow the *<!Font Definitions, ...>* declaration and precede any declaration that refers to color or pattern, such as *<!Master Diagramming Object, ...>* or *<!Master Frame, ...>*.
- *<!Revision Tracking, ...>*, if used, must precede *<!Class, ...>* declarations.
- *<!Class Defaults, ...>*, if used, must appear after the *<!Font Definitions, ...>* declaration and before the *<!Class, ...>* declarations. Class defaults are a special case. See the *<!Class Defaults, ...> Declaration* in the chapter *ASCII Format for Text* for more information.
- *<!Class, ...>*, if used, must appear after the *<!Font Definitions, ...>* and *<!Class Defaults, ...>* declarations. This declaration is used to declare all component types.
- *<!Master Frame, ...>* declarations must appear after the *<!Class Defaults, ...>* and *<!Class, ...>* declarations, since a master frame might contain components.
- *<!Master Diagramming Object, ...>* declarations must appear after *<!Master Frame, ...>* declarations, if any.
- *<!Master Table, ...>* declarations must appear after the *<!Master Frame, ...>* declarations, if any.
- *<!End Declarations>* must follow all other declarations and must precede all commands.



---

## Syntax for Interleaf ASCII Format

### General Conventions

Within the markup brackets, there must be a comma after each command word that is followed by pertinent information. Use a comma to separate one property and value from the next. For example:

```
<!Class, personal,
    Bottom Margin = 0.15 inches,
    Line Spacing = 1 lines,
    Font = F6>
```

Comments may be included. To indicate a comment within markup, insert two dashes (--) on either side of the comment text. For example:

```
<!Class, personal, --use this only for private memos--
    Bottom Margin = 1.15 inches,
    Line Spacing = 2 lines,
    Font = F6>
```

There must be no spaces between the two dashes, nor should there be special characters, such as angle brackets, in the comment text.

Blank lines, tabs, and spaces before the first component (for example, between the declarations) are ignored. If a character other than these is seen, a default component is created for the character and any that follow.

---

#### Note

You cannot place comments within diagramming data.

Within the markup brackets (< >) you may use tabs, blanks, and newlines, following the rules in *Quoting Conventions* in this chapter. Interleaf 7 does not save these tabs, blanks, and newlines if you load the document and then resave it in ASCII on your desktop.

A right angle bracket (>) alone, with no preceding <, is considered a text character. To use a left angle bracket as a text character rather than a markup indicator, you must double it (<<). For example, if you want to see the expression 2<10 in your document the corresponding markup must be 2<<10.

## Quoting Conventions

There is no harm in using quotation marks even when they are not necessary. When you are unsure whether to use quotation marks in declarations or commands, use them.

If a name contains any of the following characters, enclose the name in quotation marks:

- spaces (including thin, hairline, em, and en spaces)
- commas
- minus signs
- en dashes
- em dashes
- small bullets
- = < > @ ” ’ ! # | { } ~ \ ^ ` £ § “ ¢
- special characters—that is, any character with a hex value less than hex 20 or more than hex 7e, including Ä, Ö, ê, and similar accented characters.
- Use the <#xx> convention for special characters in quoted names. The <#xx> convention is described in the chapter *ASCII Format for Text*.

When you use the characters listed above inside any of the following entries, you must enclose the entire entry in quotation marks:

- names of components, frames, autonumber streams, or printers
- page numbers with prefixes
- autonumbers with prefixes or suffixes
- index headings
- filenames

---

**Note**

If a component name contains strings that are the same as command names or font names, such as *tab*, *F0*, or *HR*, you must enclose the entire name in quotation marks. For example, if you want to call a component *table*, you must show its name like this: `<"table">`. Otherwise, the name will be interpreted as the command that creates a table in the document. See Appendix D, *Filter Writing*, for the list of reserved names.

To preserve quotation marks, enclose quoted material in outer quotation marks. For example, if you want a component name to be *Part "A"* you must enter the following:

```
"Part ""A"""
```

---

**Note**

You can use any combination of characters to name components. However, unless you have a real need for special characters, you may find it easier to deal with the names in ASCII markup if you use only alphanumeric characters.

**“Loose Matching”**

When you open a document on the desktop, if a name indicated in a component command does not exactly match a class, Interleaf 7 tries to make a loose match by allowing upper- and lowercase letters to match and allowing for extra or missing blanks. For example, Interleaf 7 matches *Para* and *para*. This loose matching allows for recuperation from inconsistencies and typing errors when you specify component names.

## Units of Measurement

On input, when you mark up an ASCII file, you can use any of the following units of measurement:

- mm
- cm
- in (or inch, inches)
- pica (or picas)
- point (or points)
- cicero (or ciceros)
- didot (or didots)

In addition, the unit char (character) can be used in certain places. A char is defined as the size of the space character in the default font.

In this manual, inches are used as the basic unit of measurement. You may substitute any of the units listed above.

Interleaf 7 permits measurements of up to seven decimal places, with automatic rounding.

When documents are saved in ASCII, either inches or millimeters may be used. This is controlled by the value of the Lisp variable `:ascii-units`. This variable can be set on the Units sheet of the Document Properties dialog box, or with the Lisp command:

```
(tell document-name mid:set-props :ascii-units)
```

## Conventions for Specifying Properties

Property specifications in Interleaf ASCII markup may be declared as a specific numeric value, as a name or string, or as a Boolean value (i.e., *true* or *false*). For Boolean values, the following terms may be used interchangeably:

- *yes*, *true*, and *on*
- *no*, *false*, and *off*

## The Interleaf ASCII Loader and ASCII File Type

The Interleaf ASCII loader treats files without markup and files in Interleaf ASCII format differently. This section describes those differences.

### Files Without Markup

Interleaf 7 can read and interpret ASCII files that have not been marked up with the conventions this manual describes. If you read in a file that does not start with the characters `<!`, the following occurs:

- The default values established for Interleaf 7 automatically apply. These are described in detail in Appendix A, *Default Values for ASCII Format Documents*.
- For each blank line encountered in the input text, Interleaf 7 inserts a new *para* component.
- Interleaf 7 places a hard return at the end of each line of input.
- A default font is used, determined by the setting of `(ld-set-vars font-keyword ...)` and font system settings. This can be set in the Text Properties dialog box or in Lisp.
- A default dictionary is used, determined by the setting of `(sh-set-vars dictionary-keyword ...)`. This can be set in the Text Properties dialog box or in Lisp.

### Character Mapping

The ASCII loader uses a character translation table when loading ASCII files with no markup. The table converts a system's native or national character set into the Interleaf character set.

### Newlines

Newlines are treated as hard returns. Two newlines in a row (one blank line) start another instance of the prior component (the default is *para*). There must be no spaces or tabs on the blank line.

### Blank Lines

Multiple blank lines in a row are treated as one blank line.

## File Type: Icons and Extensions

The type of icon Interleaf 7 uses to represent a file on the desktop depends on how the file is named. Any file with the extension *.doc* appears as a document icon, while a file without an extension appears as a computer paper icon. Generally, document icons represent Interleaf 7 files and computer paper icons represent ASCII files on the desktop.

Interleaf 7 reserves the following filename extensions for internal use. Do not use these as extensions in filenames for files intended to be treated as documents:

- .drw            Drawer
- .fdr            Folder
- .cab            Cabinet
- .sty            Catalog
- .img            Image
- .dgm            Diagram
- .boo            Book
- .lsp            Lisp
- .lo             Lisp (compiled)
- .fas            Lisp (compiled fast format)
- .clp            Clipboard
- .pl             Printerleaf
- .ter            Terminal
- .plt            Plot
- .scn            Scan
- .pal            Palette
- .dir            Directory
- .eps            Encapsulated Postscript
- .cht            Chart
- .dct            Dictionary
- .mas            Binder

---

## Character Mapping

Interleaf 7 uses an extension of the ISO Latin/1 character set for character mapping. Upon loading, the character translation table first determines if any of the first four characters is illegal. The null character is always illegal. If the file is legal, mapping proceeds.

It may be necessary to adjust the character map table to accommodate differences in character set of a platform specific nature. The character map table is located in the file *char.map* in the *data* directory. You can edit a local copy of the system *char.map* file and create your own translation files. For example, you can map the left square bracket ( [ ) to hex code c4 (the letter *A* with an umlaut) by adding the following line to your *char.map* file:

```
| [ | \xc4 |
```

On a Digital workstation, for example, you might use the character translation table to map the few characters that do not match the ISO Latin/1 character set used by Interleaf 7. You can also use the table to map a file written in a native 7-bit character set, or to map from a raw IBM PC file.

If no *char.map* file exists, no mapping takes place. The loader assumes the characters are in the ISO Latin/1 encoding.

---

## ASCII Error Messages

Interleaf 7 includes a Lisp-driven ASCII loader message function. This function tracks the loading of ASCII text and creates a message box detailing any irregularities in the conversion. Information is kept in the message box and may be printed or filed as an ASCII document. For a full description of error warnings, see Appendix A, *ASCII Format Error Messages*.

---

## Creating and Editing ASCII Format Documents

There are several ways to create an Interleaf ASCII format document.

- Open an Interleaf 7 document and select ASCII on the **Save** dialog box on the **File** menu.
- Use an Interleaf filter or your own custom filter to convert a non-Interleaf file into Interleaf ASCII format.
- Use a text editor to create a document and manually mark it up with Interleaf ASCII format declarations and commands.
- With a text editor, mark up just the body of the document with Interleaf ASCII format commands, and use the `<!Include Declarations, ...>` command to supply the basic structure of the document. For more information about the `<!Include Declarations, ...>`, refer to the chapter *The Include Commands*.

### Information Discarded by Interleaf ASCII

Certain information is not saved when Interleaf binary (“Fast”) documents are resaved in Interleaf ASCII format. This includes

- discretionary hyphens inserted by the user; elimination of these hyphens might change line endings and, less frequently, pagination
- “state” information about such things as text caret position and display of frame anchors and hard returns; this has no effect on the content of the document
- in rare cases, a property that violates allowable parameters will be changed; for example, a margin that specifies a negative width column

Documents with frozen composition—that is, documents with the Composition Freeze property set to **Yes** on the Custom Page property sheet—cannot be saved in ASCII format.



# *ASCII Format for Text*

# 2

This chapter describes the Interleaf ASCII format for text. Interleaf ASCII format, or markup, consists of declarations and commands; both declarations and commands describe properties. This chapter describes the properties for each declaration and command.

The order of commands and declarations in this chapter corresponds to the internal structure of an Interleaf document. In addition, default values for markup are indicated in boldface type.

For more information about property defaults, refer to Appendix A, *Default Values for ASCII Format Documents*. For special cases, refer to Appendix D, *Filter Writing*.

---

**Note**

Starting with Release 5.2, components and graphics objects can have Interleaf Lisp statements attached upon output. Although method storage is not fully supported, some changes will affect output. Changes affecting text are described in `<!Class, ...> Declaration` — *Declaration for Components* in this chapter.

## Declarations

The order in which Interleaf ASCII declarations are assembled for markup is important. For details, refer to *Required Order of Declarations* in the chapter *ASCII Format Basics*.

### <!OPS, ...> Declaration

Interleaf strongly recommends that you use the `<!OPS, ...>` declaration as the first statement in your file. This guarantees the proper defaults are installed. If *Version* is not specified, the latest version is assumed.

```
<!OPS, Version = n.n>
```

For the current release of Interleaf, the Version number is 8.2.

### Metric option

To create a metric A4 document with the default values listed below, include the specification `Style = Metric` inside the `<!OPS, ...>` declaration.

The `Style = Metric` specification automatically generates these values under the `<!Page, ...>` declaration:

```
Top Margin = 25 MM,  
Bottom Margin = 28 MM,  
Left Margin = 35 MM,  
Right Margin = 35 MM,  
Inner Margin = 25 MM,  
Outer Margin = 25 MM,  
Page Width = 210 MM,  
Page Height = 297 MM,
```

The `Style = Metric` specification also generates the following `<!Class, ...>` declaration component value:

```
Bottom Margin = 3.5 MM
```

### <!Page Number Stream, ...> Declaration

This declaration permits simultaneous tracking of numbering sequences, such as pages within sections and sections within books. Different page streams can be created for a single document and can vary with respect to style, prefix, and starting number.

---

**Note**

A-page numbering is controlled through this declaration, not through the `<!Document, ...>` declaration as in some older releases.

The following example shows the properties that are declared under `<!Page Number Stream, ...>`:

```
<!Page Number Stream,
    Name = string,
    Starting Page # = n,
    Page # Prefix = string,
    Page # Prefix Two = string,
    Page # Style = Arabic, Lower Case Roman, Upper Case
        Roman, Upper Case Alpha, Lower Case Alpha
    A-Page # Style = Arabic, Lower Case Roman, Upper
Case      Roman, Upper Case Alpha, Lower Case Alpha,
    A-Page # Prefix = string,
    Maximum Page # = n,
    Prefix Stream = string,
    Begin with = odd, even, any
```

If no page stream information is specified, a default Page Stream Name is generated by the ASCII loader. The default specification is:

```
Name = page
```

If a name is specified for the page stream, it may not exceed 19 characters in length.

The *Page # Prefix* property determines a prefix string, such as *Page*, to precede the page number on the page. If *Page # Prefix Two* is used, it creates a second string that appears after the first, preceding the number.

With the *Page # Style* property, you can specify a variety of styles for the page number: Arabic, Lowercase Roman, Uppercase Roman, Lowercase Alpha, or Uppercase Alpha.

*A-Page # Style* property specifies the style in which A-page information appears. The options are identical to those for the *Page # Style*.

*A-Page # Prefix* property determines a prefix string, which occurs on each A-Page. This name may not exceed nine characters in length.

*Maximum Page #* specifies the page number of the document after which A-pages are added. If *Maximum Page #* is set to the value of 5, every page after 5 is numbered 5a, 5b, 5c, and so on.

*Prefix Stream* lists the name of the page prefix autonumber stream. For more information about autonumber streams, refer to *<!Autonumber Stream, ...> Declaration* in this chapter.

## <!Document, ...> Declaration

The markup for the *<!Document, ...>* declaration is as follows:

```
<!Document,
    Header Page = Yes/No,
    Double Sided = Yes/No,
    Manual Sheet Feed = Yes/No,
    Print Rev Bars = Yes/No,
    Print Strikes = Yes/No,
    Print Underlines = Yes/No,
    Print Deletion Marks = Yes/No,
    Underline at Descender = Yes/No,
    Final Output Device = string,
    Default Printer = string,
    Orientation Inverted = Yes/No,
    Measurement Unit = inches/picas/points/ciceros/MM/
        picas:points/ciceros:didots,
    Line Spacing Unit = inches/picas/points/ciceros/MM
        didots/picas:points/ciceros:didots,
    ASCII Unit = inches/MM,
    Font Unit = points/didots/MM,
    Measurement Precision = n units, [2]
    Float Precision = n units, [2]
    Points Precision = n units, [1]
    Spot Color Separation = Off/Screened/Solid,
    Component Bar Width = x inches, [.906]
    Zoom = x [1],
    Facing Pages = Yes/No,
    Default Page Stream Name = page [or default name]>
```

### Header Page

The *Header Page* property determines whether a header page prints at the front of the document. The header page prints such information as the name of the user printing the document, and the time and date of printing.

### Double Sided

The *Double Sided* property determines whether the document prints on one side of the paper or on both sides.

<b>Manual Sheet Feed</b>	The <i>Manual Sheet Feed</i> property determines whether the paper is to be fed into the printer automatically or manually.
<b>Print Rev Bars</b>	The <i>Print Rev Bars</i> property determines whether displayed revision bars are printed.
<b>Print Strikes</b>	The <i>Print Strikes</i> property determines whether displayed strikethrough lines are printed.
<b>Underline</b>	<p>The <i>Print Underlines</i> property determines whether displayed underlines are printed.</p> <p>The <i>Underline at Descender</i> property determines whether underlines are drawn at the descender or at the baseline.</p>
<b>Print Deletion Marks</b>	The <i>Print Deletion Marks</i> property determines whether marks indicating deleted material are printed. These marks are records of deletion that are kept when Revision Control has been activated in a document.
<b>Final Output Device</b>	The <i>Final Output Device</i> property determines the type of output device to which your document is sent; you specify a device type such as cx or ps.
<b>Default Printer</b>	The <i>Default Printer</i> property determines the specific device to which your document is sent; you specify a device such as nearest device, or the actual name of the device.
<b>Orientation Inverted</b>	If set to Yes, the <i>Orientation Inverted</i> property reverses the displayed page orientation for printing. For example, a page displayed as a portrait page prints as a landscape page when <i>Orientation Inverted</i> is set to Yes.
<b>Measurement Unit</b>	The <i>Measurement Unit</i> property determines the display unit for numerical values in most dialog box fields. For example, if the value for <i>Measurement Unit</i> is picas, the system displays numerical values in picas.
<b>Line Spacing Unit</b>	The spacing between lines of text can be set in any of the measurement units shown in the markup example.

.....

<b>ASCII Unit</b>	The <i>ASCII Unit</i> property affects ASCII output. If <i>MM</i> is declared, all output is metric. If you declare inches, they are the unit. No other units are supported.
<b>Font Unit</b>	Fonts may be specified in any of the units of measurement shown in the markup example.
<b>Zoom</b>	The <i>Zoom</i> property determines the degree of magnification seen by the user on the screen. Settings for <i>Zoom</i> range from .25 (smallest possible) to 16 (largest possible).
<b>Facing Pages</b>	If <i>Facing Pages</i> is declared, the document opens showing left and right pages opposite each other. The amount visible for each page depends on the setting for <i>Zoom</i> .
<b>Spot Color Separation</b>	The <i>Spot Color Separation</i> property determines how pages are printed for color separation. A value of Off prints all color on a single page of paper. A value of Solid prints each color (including different percentages of the same color) on a separate page. A value of Screened is the same as Solid except that different percentages of the same colors print on the same page.
<b>Component Bar Width</b>	The default width of the component bar is .906 inches. In adjusting component bar width, do not declare too narrow a width. This truncates component names on the screen.
<b>Default Page Stream Name</b>	The default name of a page stream is always declared under the <i>&lt;!Document, ...&gt;</i> declaration.

---

**Note**

All A-page declarations for prefix are now handled in the *<!Page Number Stream, ...>* Declaration. The procedure used varies from some releases.

**<!Page, ...> Declaration**

The properties and values for the `<!Page, ...>` declaration reflect most of the selections on the Page Properties dialog box.

The markup for the `<!Page, ...>` declaration is as follows:

```
<!Page,
    Columns = n, [1]
    Gutter = x inches,
    Height = x inches, [11]
    Width = x inches, [8.5]
    Top Margin = x inches, [1]
    Bottom Margin = x inches, [1.1]
    Left Margin = x inches, [1.4]
    Right Margin = x inches, [1.4]
    Inner Margin = x inches, [1]
    Outer Margin = x inches, [1]
    Margins = Left/Right|Inner/Outer,
    *Balance Columns = On/Off,
    Vert. Just. = On/Off,
    Margin Stretch = n, [200%]
    Margin Shrink = n, [50%]
    *Frame Margin Stretch = n, [10%]
    Feathering = On/Off,
    Max. Feathering = n, [8%]
    *Depth At Page Break = n, [95%]
    *Depth No Page Break = n, [90%]
    *First Page = Left/Right,
    *Bleed = Yes/No,
    Hyphenation = On/Off,
    Consecutive Hyphens = Any/1/2/3/4,
    *Allow Break After Hyphen = Yes/No,
    Revision Bar Placement = Left/Right/Automatic,
    *Turned Pages = 0/90/270,
    *Frozen Number Streams = Yes/No
    Vertical Margins = Add/Maximum/Bottom/Top
    vertical text = Yes/No
    Baseline to Baseline = Yes/No>
```

In this markup, items marked with an asterisk (\*) are not allowed in a `<!Page, ...>` declaration for a microdocument.

**Height  
and Width**

The *Height* and *Width* properties determine page size. When width is output for microdocuments whose size is computed relative to the page or contents, it is for information only and is recomputed when the document is loaded. When metric defaults are in use some of these defaults are different. See Appendix A, *Default Values for ASCII Format Documents*.

**Margins**

Together, page and component margins must allow at least 0.5 inch of space, horizontally and vertically, for text per column, except in microdocuments.

**Columns**

The *Columns* property determines the number of columns on the page.

**Gutter**

The *Gutter* property determines the amount of white space between columns in a multicolumn document. You can specify any positive value for the *Gutter* property.

**Balance Columns**

The *Balance Columns* property determines whether Interleaf 7 balances columns on all pages of a document. When *Balance Columns* is On, the software tries to match column depth as closely as possible on all pages using vertical justification. When *Balance Columns* is Off, text must reach the bottom margin before it moves into the next column on the last page of the document.

**Vertical  
Justification**

*Vert. Just.*, *Margin Stretch*, *Margin Shrink*, *Frame Margin Stretch*, *Feathering*, *Max. Feathering*, *Vert. Just. Pages*, *Depth At Page Break*, and *Depth No Page Break* are vertical justification properties. Vertical justification adjusts the vertical spacing in a document by expanding or compressing the white space between components, lines, and frames.

The default setting for *Vert. Just.* is On. If you do not want Interleaf 7 to adjust vertical spacing at all, the value for the *Vert. Just.* property is Off.

The values for *Margin Stretch*, *Margin Shrink*, *Frame Margin Stretch*, *Max. Feathering*, *Depth At Page Break*, and *Depth No Page Break* are percentages.

*Margin Stretch*, *Margin Shrink*, *Frame Margin Stretch*, *Feathering*, and *Max. Feathering* determine how much Interleaf 7 can stretch and shrink white space to achieve column and page justification.

.....



To achieve column justification, set at least some of these options to On and make sure *Vert. Just.* is On. To achieve page justification, you must also turn on the *Vert. Just. Pages* option. The settings for all options are effective only when *Vert. Just.* is On.

The default value for *Margin Stretch* is 200%. With this setting, Interleaf 7 stretches the margins between components up to 200% to achieve vertical justification. You can set *Margin Stretch* from 0% (off) to 400%.

The default value for *Margin Shrink* is 50%. With this setting, the software compresses the spaces between components by as much as half to achieve vertical justification. You can enter a setting of 0% to 100%.

The default value for *Frame Margin Stretch* is 10%. This means Interleaf 7 can increase the space above and below frames by as much as 10% of the height of the frame to achieve vertical justification. You can set *Frame Margin Stretch* from 0% to 100%.

The *Max. Feathering* property determines the maximum amount of space the software adds between lines to achieve vertical justification. The default value for *Max Feathering* is 8% when *Feathering* is On. You can change this value to any positive percentage.

When *Vert. Just. Pages* is On, you can specify how full a page must be before it is vertically justified.

The default setting for *Depth At Page Break* is 95%. If you want Interleaf 7 to try to justify the contents when the page is less than 95 percent full, change the setting to a lower percentage.

The default setting for *Depth No Page Break* is 90%. You can increase or decrease this setting.

## **Bleed**

If the *Bleed* property is set to Yes, the margins of header and footer frames expand to the full width of the page.

## **Hyphenation**

The *Hyphenation* property determines whether hyphenation is on or off for the entire document. *Hyphenation* must be On for the other hyphenation properties to be in effect.

The *Consecutive Hyphens* property determines how many consecutive lines can end with a hyphenated word.

.....

The *Allow Break After Hyphen* property determines whether you can break a page at a hyphenated word.

---

---

**Note**

Setting *Allow Break After Hyphen* to No is not fully supported.

**Revision Bar Placement**

The *Revision Bar Placement* property determines where Interleaf 7 places revision bars. The default is Left, which places revision bars in the left margin of the page. Right places revision bars in the right margin of the page.

Specifying Automatic as the value for *Revision Bar Placement* produces the following results:

- Single-sided, single-column layouts continue to have revision bars on the left.
- Double-sided, single-column layouts have revision bars on the outside of the page (on the right for the right-hand page and on the left for the left-hand page).
- Single- and double-sided, two-column layouts have revision bars on the outside of each column (on the right for the right column and on the left for the left column).

For more information about revision bars and revision tracking, refer to *<!Revision Tracking, ...> Declaration* in this chapter.

**Turned Pages**

The *Turned Pages* property rotates the contents of a page between headers and footers; header and footer information keeps its original orientation unless you also turn the headers and footers.

To turn a page clockwise, declare the value of the *Turned Pages* property to be 90; to turn a page counterclockwise, declare the value of the *Turned Pages* property to be 270.

Pages rotated clockwise have the header on the right margin of the page and the footer on the left margin of the page; pages rotated counterclockwise have the header on the left and the footer on the right.

You declare the header and footer rotation as clockwise or counterclockwise by specifying 90 or 270, respectively, for the *Rotation* property in the *<Page Header>* and *<Page Footer>* commands.

**Vertical Margins**

The *Vertical Margins* property determines the way distance between two components is calculated. If Top is selected, the top margin of the lower component determines the distance between the two. With Bottom, this distance is determined by the bottom margin of the upper component. The default is Add, which adds the top and bottom margins and applies this distance between any two components. If Maximum is selected, the larger of the two (top and bottom) is the distance between components.

**Baseline to Baseline**

When *Baseline to Baseline* is set to Yes, spacing between components is determined as the distance from the baseline of the last line of the top component to the baseline of the first line of the next component.

**Frozen Number Streams**

If the *Frozen Number Streams* property is set to Yes, the values of autonumber tokens and properties and the properties of autonumber streams do not change when tokens are added to or removed from the stream.

---

**Note**

You cannot change the properties of any autonumber streamf when it is frozen. This restriction is new with Interleaf 5.4J and Interleaf 7.

**<!Font Definitions, ...> or <!Fonts, ...> Declaration**

The basic form of the font declaration is *<!Font Definitions, ...>* followed by a list of the fonts used in the document.

```
<!Font Definitions,
    Fn = font name,
    Fm = font name,
    ...>
```

In markup, *<!Font Definitions, ...>* and *<!Fonts, ...>* are synonymous. Throughout the rest of the text portion of the marked up file, *<Fn>* appears as a concise reference to the font defined in the *<!Font Definitions, ...>* declaration. Refer to the chapter *ASCII Markup for Graphics Objects* for font conventions inside graphics objects.

You can specify font names, sizes, and faces in flexible order. Case is not considered. For example, **Times 10 Bold** and **times bold 10** are equivalent.

Font names are not enclosed in quotes even if they contain spaces. For example, you can have New York 10, but not "New York" 10. Font names may only contain standard alphanumeric characters; special characters and symbols are not permitted.

An example of a simple font declaration is

```
<!Font Definitions,
      F2 = Times 10,
      F3 = Times 10 Italic>
```

For files that have been marked up for use with Interleaf, the ASCII loader attempts the closest match if the font you specify is not available. The presence in a document of font definitions for fonts not used in the document has no effect on the document.

Font size is in points and may be declared for input to one decimal place between the sizes 2 and 200. The following declarations are examples of legal font sizes:

- F3 = Thames 13.7,
- F4 = Courier 15.0,
- F16 = Thames 28.3,
- F17 = Courier 32.5,
- F23 = Thames 110.3,

When you zoom a document, a larger or smaller font must be loaded to display the document. If you save a zoomed document in ASCII markup, the font size of the zoomed font is recorded in the file. A font you declared to be Courier 15 might appear as Courier 15.7. When the zoom is reset to 100%, the original value for font size is restored.

---

**Note**

Font numbers are arbitrarily assigned place-holders for the fonts used in the document. A different version of the software may have a font table that is larger or smaller, so font F4 may not Courier 15 in an ASCII format document dumped in that environment. A filter should parse the full font declaration, not depend on font numbers.

**<!Color Definitions, ...> or <!Colors, ...> Declaration**

The `<!Color Definitions, ...>` or `<!Colors, ...>` declaration defines the set of colors the document can use. This declaration associates each color with a unique code number. The code numbers refer to colors in the document markup. Code numbers can range from 0 to 255. C0 is usually white. In markup, `<!Color Definitions, ...>` and `<!Colors, ...>` are synonymous.

The `<!Colors, ...>` declaration usually follows the font declaration in the markup for a document, although it can appear anywhere before the first instance of a color in the document. There should be only one `<!Colors, ...>` declaration in a document.

Every document has a color palette. If no `<!Colors, ...>` declaration appears in a document, the default color palette is used. When a document with the default color palette is saved, the color palette declaration is omitted from the output ASCII markup.

When you create colors interactively, definitions for these colors are dumped in ASCII and added to the `<!Colors, ...>` declaration.

The generic format for the color declaration is:

```
<!Colors,
    Cn = cyan, magenta, yellow, black
    Cm = cyan, magenta, yellow, black
    ...>
```

The following is the default `<Colors, ...>` declaration (all the values are shades of gray):

```
<!Color Definitions,
    C0 = 0, 0, 0, 0,
    C1 = 0, 0, 0, 3.125,
    C2 = 0, 0, 0, 6.25,
    C3 = 0, 0, 0, 12.5,
    C4 = 0, 0, 0, 25,
    C5 = 0, 0, 0, 50.001,
    C6 = 0, 0, 0, 75.001,
    C7 = 0, 0, 0, 100>
```

The *C* preceding the code number can be either uppercase or lowercase. If the letter is uppercase, its corresponding color is displayed on the color popup menus. If the letter is lowercase, its color is not displayed on the color popup menus, but objects in the document can still use this color. An example of this markup follows. The color *C8* or *c8* was created using the color palette and has the following markup:

```
C8 = 50.277, 24.999, 69.999, 12.776
```

or

```
c8 = 50.277, 24.999, 69.999, 12.776
```

The identifying numbers that follow the letter *C* correspond to the numbers for the default colors on the popup menus. The numbers do not appear on the popup menus, but you can see the default colors by looking at the Props Fill Color submenu for a default document. Figure 2-1 shows the default color numbers and the colors they produce within Interleaf 7.

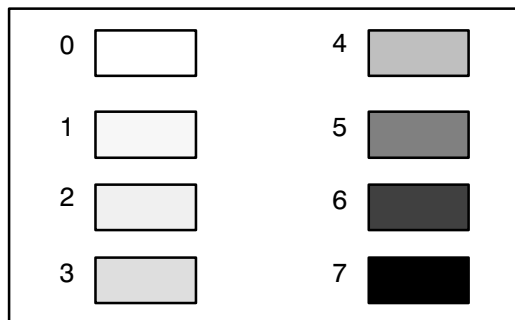


Figure 2-1. Default colors and values.

The values for each color in the `<!Colors, ...>` declaration are percentages of cyan (C), magenta (M), yellow (Y), and black (B). Each color is defined in terms of percentages of these primary colors. These percentages are represented by floating-point numbers from 0 to 100. For non-color (black-and-white) markup, all relevant color information can be placed in the fourth column. The columns corresponding to C, M, and Y must each be given the value of zero in this case (as shown in the example markup).

## <!Pattern Definitions, ...> or <!Patterns, ...> Declaration

The *<!Pattern Definitions, ...>* or *<!Patterns, ...>* declaration defines the set of patterns that can be used by the document and associates each pattern with a unique code number. The code numbers are used when referring to patterns in the document markup. Code numbers can range from 0 to 255. In markup, *<!Pattern Definitions, ...>* and *<!Patterns, ...>* are synonymous.

The *<!Patterns, ...>* declaration usually follows the *<!Fonts, ...>* and *<!Colors, ...>* declarations in the markup for a document, although it can appear anywhere before the first instance of a pattern in the document. There should be only one *<!Patterns, ...>* declaration in a document.

The generic format for the *<!Patterns, ...>* declaration is:

```
<!Pattern Definitions,
    P0 = bitmap,
    P1 = bitmap,
    ...>
```

The **bitmap** in a pattern definition is a string of hexadecimal digits that represent the bits in the pattern. The first four digits represent the 16 pixels in the first scan line of the pattern. The most significant bit corresponds to the left-most pixel. A bit value of 1 indicates that the corresponding pixel is part of the pattern and is colored. A bit value of 0 indicates that the pixel is white. The next four digits represent the next scan line, and so on. There are 16 scan lines per pattern.

When you save a document with the default pattern palette, the pattern declaration is omitted from the output ASCII markup. The following is the default pattern definition:

```
<!Pattern Definitions,
P0 = fdffd0000dfdfdfdfdfdf0000fdfdffdfdfdf0000dfdfdfdfdfdf0000fdfdffdf,
P1 = 060628286e6e8282eccec2828c0c08282060628286e6e8282eccec2828c0c08282,
P2 = 82820606080828286e6e88888080e0e082820606080828286e6e88888080e0e0,
P3 = 0c0c0c0c0c0c0c0c0cfcfcfcfc000000000c0c0c0c0c0c0c0c0cfcfcfcfc00000000,
P4 = 08080c0c0c0c1c1cfcfc78780000000008080c0c0c0c1c1cfcfc787800000000,
P5 = ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff,
P6 = c0c06060303018180c0c060603038181c0c06060303018180c0c060603038181,
P7 = 8181030306060c0c181830306060c0c08181030306060c0c181830306060c0c0,
P8 = ffffffffffffffffff0000000000000000fffffffffffffffff0000000000000000,
P9 = f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0,
P10 = f0f078783c3c1e1e0f0f8787c3c3e1e1f0f078783c3c1e1e0f0f8787c3c3e1e1,
P11 = 0f0fle1e3c3c7878f0f0e1e1c3c38787f0f0fle1e3c3c7878f0f0e1e1c3c38787>
```

The pattern definition defines the pattern at screen resolution (75 dots per inch). When the pattern is printed, it is automatically scaled to the resolution of the printer (for example, 300 dpi for most laser printers).

Figure 2-2 shows the default patterns and the pattern numbers they produce. You can see the default patterns by looking at the Props Fill Pattern submenu for a default document. The number values do not appear on the popup menus.

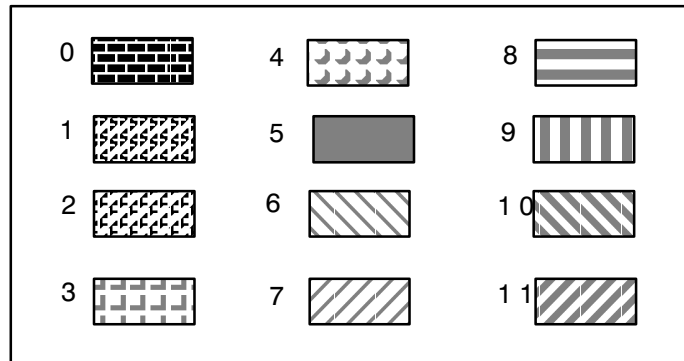


Figure 2-2. Default patterns and values.



**<!Revision Tracking, ...> Declaration**

Revision tracking lets you create, view, and manage different versions of the same document. This feature supports both parallel and sequential editing and can record additions or deletions to material contained in components, named graphics objects, tables, and inline components. Information about change is recorded in the form of attribute values. For detailed information about revision tracking, consult the *Interleaf 7 User's Guide*.

Markup for revision tracking can be complex. The following example has been simplified to demonstrate the basic configuration. It shows the markup for an editing episode named *dante*.

**Revision Tracking  
Declaration****RevLevel  
Declaration****RevVersion  
Declaration**

```
<!Revision Tracking,
  Revision Tracking = On,
  Current Version = "dantever",
  Current Level = "dante",
  <!RevLevel,"dante", 0,
  Insert Text Props = @attr,
  Delete Text Props = @attr,
  Insert Text Color = @attr,
  Delete Text Color = @attr,
  Inactive Insert Props = @attr,
  Inactive Delete Props = @attr,
  @"<<RevEditId" = "dante">
  <!RevVersion,"<<Base">">
  <!RevVersion,"dante",
  Contents =      "dante",
  Active Level =  "dante">
>
```

*<!Revision Tracking,...>*, the opening declaration, encompasses both *<!RevLevel,...>* (which specifies the edit under way or previously undertaken), and *<!RevVersion,...>*, which indicates Contents and Active Level of each individual version, assuming a default Base as the starting point.

There are several optional attributes that can be specified with properties listed under *<!RevLevel,...>*; these can be set to show alterations to data.

## <!Autonumber Stream, ...> Declaration

<!Autonumber Stream, ...> declarations must appear before the <!Class Defaults, ...> and <!Class, ...> declarations.

The basic form of the autonumber declaration is <!Autonumber Stream, ... followed by the name of the autonumber stream, the number of levels in the stream, and the closing angle bracket (>).

```
<!Autonumber Stream, name, n,  
    Level m Symbol Type = Arabic/Upper Roman/  
        Lower Roman/Upper Alpha/Lower Alpha,  
    Level m Prefix = string,  
    Level m Suffix = string [.] ,  
    Level m Starting Value = n [1] ,  
    Level m Trail = Yes/No ,  
    Level m Show = Yes/No>
```

The *n* after *name* represents the number of levels, which you must specify. If a value for *n* is not specified, Interleaf 7 assumes 20 levels.

Trail represents Last Level Only on the Autonumber Stream Properties dialog box.

You can define stream properties between the number of levels (*n*) and the closing angle bracket by using entries comparable to those on the Autonumber Stream Properties dialog box.

The other entries for each stream level through *n* must be declared only if they are different from the default values. For a list of the default values, refer to Appendix A, *Default Values for ASCII Format Documents*.

## <!Class Defaults, ...> Declaration

Certain values for component properties are used frequently. For example, many components have a left margin of 0 inches. For convenience, Interleaf ASCII format assumes a number of default values for component properties.

When marking up a document, you can override any of the defaults and declare a different property value to be used throughout the document by inserting a *<!Class Defaults, ...>* declaration. For example, if you want the text of a document to be Times 10 italic instead of Times 10 roman, enter:

```
<!Class Defaults,
      Font = F5>
```

This font must have been defined in the *<!Font Definitions, ...>* declaration, which must precede the *<!Class Defaults, ...>* declaration.

In the *<!Class Defaults, ...>* declaration, you can specify a value for any component property except tabs. For more information about the properties and values used in the *<!Class Defaults, ...>* declaration, refer to *<!Class, ...> Declaration* in this chapter.

### Fill

The *Fill* property controls the treatment of newlines in the input document. The component property *Fill = off* inserts a hard return at the end of each line. This is useful for components containing tabular material.

When *Fill = blank*, the loader inserts a blank at the end of each input line if no blank was there and if it is not the last line of a component. This is useful for documents that have been prepared with word processors, or with editors that do not ensure blanks at ends of lines. For more detailed information, see Appendix D, *Filter Writing*.

---

### Note

If your document has *Fill = off* in its declarations, and also has an *<!Include Declarations, ...>*, *Fill* is overridden by the declarations loaded in from the included file. You must be sure that the *Fill = off* statement is in the included file to take effect.

## <!Class, ...> Declaration — Declaration for Components

A component `<!Class, ...>` declaration defines the name and properties of a master component. This provides the ASCII loader with property values to use in loading components of a given name. Most of the information specified in the component `<!Class, ...>` declaration is identical to the information on the Component Properties dialog box in Interleaf 7. When a property has the default value, however, it is not listed in the ASCII format component class declaration.

When a document is saved, the properties assigned to each component class are compared with the properties of the master component for that class. There are two exceptions:

- The properties of a frame within a component are always checked against the global defaults for that frame.
- When a master component has an inline component in a microdocument in a frame as part of its contents, the master for that inline component might not have been dumped yet. So the properties of a component in a microdocument in a frame or a master row in a master table are not checked against their master, they are checked against the global defaults.

User-defined attributes must be specified each time they occur. In Interleaf ASCII format, a user-defined attribute/value specification on the component master is not created automatically for instances of that master. If an instance has user-defined attributes, you must specify them with the instance even if you already specified them on the master in the `<!Class, ...>` declaration. For more information, refer to *User-Defined Attributes* in this chapter.

---

**Note**

It is an error to have two or more `<!Class, ...>` declarations specifying the same component name.

The markup for the `<!Class, ...>` declaration is as follows:

```
<!Class, component name,
    Character Units = Yes/No,
    Top Margin = x inches, [0]
    Bottom Margin = x inches, [0.14]
    Left Margin = x inches, [0]
    Right Margin = x inches, [0]
    First Indent = x inches, [0]
    Indent Count = n, [1]
    Line Spacing = x lines or x points, [1.31 lines]
    Alignment = Left/Right/Both/Center/Inner/Outer,
    Font = Fn, [Times 10]
    Begin New Page = Yes/No,
    Begin New Column = Yes/No,
    Straddle = Yes/No,
    Orphan Control = 1-16, [2]
    Widow Control = 1-16, [2]
    Allow Page Break Before = Yes/No,
    Allow Page Break Within = Yes/No,
    Allow Page Break After = Yes/No,
    Hyphenation = On/Off/Normal/Full/0-10, [5]
    Composition = Optimum/Overset,
    Track Kerning = n, [0]
    Track Kern Spaces = Yes/No,
    Wordspace Min = x,
    Wordspace Nom = x,
    Wordspace Max = x,
    Wordspacing = Default,
    Letterspacing = Yes/No,
    Letterspace Max = x,
    TOC Doc Name = name,
    TOC Page Stream = name
    Left Tab = x inches,
    Right Tab = x inches,
    Tab Origin = Column/Margin
    Decimal Tab = x inches,
    Decimal Char = "<char>"
    Numeric Tab = x inches
    Left Profile = start; count; indent,
    Right Profile = start; count; indent
    Profiling = None,
    Contents = Private/Prefix/Shared/Prefix Shared
    Lisp = "actual Lisp code">
```

## Name

The *component name* can be up to 19 characters long. the component name can be any combination of letters, numbers, and other characters. However, unless you have a particular need for other characters, it is less confusing if you use only alphanumeric characters. Uppercase and lowercase letters are considered different characters. Refer to *Quoting Conventions* in the chapter *ASCII Format Basics* for guidelines on using non-alphanumeric characters.

## Margins

The left and right component margins are the distances from the page margins to the left and right edges of the component. When the component margins are set to 0 inches, the component is as wide as the area between the left and right page margins.

Top component margins have no effect at the top of a page; bottom component margins have no effect at the bottom of a page.

Together, page and component margins must allow at least 0.5 inch of space for text per column, except in microdocuments. *Margin* and *Indent* settings must be consistent. For example, negative indents must not overlap the page edge, and positive indents must not exceed the right margin.

## Indent

The *Initial Indent* property determines the amount of indentation for the first line or group of lines in the component.

Use the *Indent Count* property to specify the number of lines to be indented by the amount that you specified in the *Initial Indent* property.

## Alignment

The *Alignment* property determines the horizontal alignment of text in a component.

- A value of Left aligns the text at the left component margin and leaves a ragged-right component margin.
- A value of Right justifies the text at the right component margin and leaves a ragged-left component margin.
- A value of Both (justified) causes text to be justified between the left and right component margins. The word spacing expands to fill each line. This is the default.

- A value of *Inner* aligns the text at the inner margin (the left margin on a right-hand page, for example), leaving the other margin ragged.
- A value of *Outer* aligns the text at the outer margin (the right margin on a right-hand page, for example), leaving the other margin ragged.
- A value of *Center* centers each line of the text between the left and right component margins. Make sure the *Initial Indent* is set to 0.

If no value is entered (default case), text is justified between the left and right component margins. Word spacing expands to fill each line.

If a component's alignment is *Centered*, *Right*, or *outer*, you cannot insert tab characters in the component.

## Font

You usually declare a *Font* for every class. The font properties are *Family*, *Size*, *Bold*, and *Italic*. These settings determine the default font for text in the component. The default is controlled by the value of `(ld-set-vars font-keyword)`.

## Begin New Page

The *Begin New Page* property forces the component to begin on a new page. In a multicolumn document, *Begin New Column* forces the component to begin in a new column.

## Widow and Orphan Control

In a single-column document, the *Orphan Control* and *Widow Control* settings determine how many lines of a component can appear at the top and bottom of a page. In a multicolumn document, these settings determine how many lines can appear at the top and bottom of a column. The default setting for *Orphan Control* and *Widow Control* is 2. You can set a value from 1 to 16. Column balancing may override widow/orphan control settings to achieve the balance.

---

### Note

In the authoring user interface, *Widow Control* and *Orphan Control* have been renamed *Minimum Lines Before Break* and *Minimum Lines After Break*.

## Allow Page Break

When you set the *Allow Break Within* property to Yes, Interleaf 7 can put part of the component on one page and part of it on the next page as long as this does not violate the widow/orphan settings for the component. If you set *Allow Break Within* to No, *Interleaf 7* does not, as a rule, break the component across page boundaries. The default value is Yes.

In multicolumn documents, the *Allow Break Within* property also determines whether a component is allowed to break across columns on the same page.

When *Allow Break After* is set to No, Interleaf 7 does not permit a page break between this component and the following component. The default value for *Allow Break After* is Yes.

In multicolumn documents, *Allow Break After* also determines whether a column break is permitted between this component and the following one.

---

### Note

If the page break settings are too restrictive, Interleaf may override them to compose the page.

## Straddle

The *Straddle* property determines whether a component extends horizontally across the entire text area of a multicolumn document.

The default setting for *Straddle* is No. With this setting, a component assumes the column width. If Yes is specified, the component spans the entire width of the text area (page width minus margins). The *Straddle* setting has no effect on components in a single-column document. A component that is Soft Straddled does not show *Straddle = Yes* in ASCII markup.

## Composition

The values you can set for the *Composition* property are Optimum and Overset. The Optimum value specifies a uniform word space as close as possible to the optimum word space for the line. Overset puts as many words on a line as possible without going under the minimum word space.



## Spacing

The spacing properties you can specify on the `<!/Class, ...>` declaration or on the `<component, ...>` command are as follows:

```
Line Spacing = x lines or x points,
Track Kern Spaces = Yes/No,
Wordspace Min = x,
Wordspace Nom = x,
Wordspace Max = x,
Wordspacing = Default,
Letterspacing = Yes/No,
Letterspace Max = x,
```

*Line Spacing* determines the distance between the lines of type within a component. It does not affect the amount of space before the first line in a component. *Line Spacing* cannot be less than -1.0 lines (-12 points). You get relative line spacing by specifying “lines.” To get absolute line spacing, specify “points” or another absolute unit of measurement.

The value for the *Wordspace* properties is a multiple of the natural word space. For example, *Wordspace Min* = .4 means 0.4 multiplied by the natural word space. The spacing properties are on the Custom sheet of the Component Properties dialog box. *Letterspacing Max* is determined with the same procedure.

You can also specify *Wordspacing* = *Default* when there are word spacing specifications on the master but not on the particular instance.

## TOC Doc Name

TOC stands for *Table Of Contents*. In the `<!/Class, ...>` declaration, you must specify a *TOC Doc Name* property for components you want included in the table of contents or a related document, such as a figure list. For example, *TOC Doc Name* = *TOC*.

When a TOC document is created, Interleaf 7 takes the first component in the document with the *TOC Doc Name* property, converts it to a TOC component (for example, *chapter1TOC*), and makes it Times 10-point bold. All other components with the same name are made Times 10-point bold. The remaining components with the *TOC Doc Name* entry are made Times 10-point roman in the TOC document.

**Tab**

There are a variety of component *Tab* properties. In the following Component Tab Properties examples, all tab stops are set at 1-inch increments:

```
Left Tab = 0/1/2/3/4/5/6/7 Inches,
Right Tab = 0/1*7 Inches,
Center Tab = 1*7 Inches, Numeric Tab = 1*7 Inches,
Decimal Tab = 1*7 Inches,
Decimal Char = "$",
```

The *Decimal Character* property causes tab stops to align on any specific character you determine instead of a decimal point. The character appears only if *Decimal Tab* has been specified.

Specify *Numeric Tab* when you want columns of numbers set with tab stops to align on the last numeric character, even though other material may follow. For example, *\$(.02)* and *1.90%* are aligned using the 2 and the 0; other characters are ignored.

A slash (/) delineates one setting from the next.

An asterisk (\*) after a number in a tab stop setting indicates evenly spaced increments. The number immediately preceding the asterisk is the amount of the increment, and the number after the asterisk is the number of increments. For example, if you set *Left Tab = 1/.5\*3* inches, tab stops start at 1, and 0.5 is added to the previous setting three times. This specification gives you left tab stops at 1, 1.5, 2, and 2.5 inches.

If there is nothing in front of the number preceding the asterisk, the first tab stop is set at 0. For example, if you set *Left Tab = .5\*3* inches, the first tab stop is 0 and 0.5 is added to the previous setting three times. The settings are at 0, 0.5, 1, and 1.5 inches.

If you do not specify any component tab property, Interleaf uses the default values.

If you do not want any tabs in the component (either in a component class declaration or in the command for a particular component), set the *Left Tab* property without a value. For example:

```
Left Tab = ,
```

The comma indicates that the entry is empty; you must use it to avoid loading the default tab stops.

## Shaping

You can use the *Left Profile* and *Right Profile* properties for shaping the text in a component. The generic form of these properties is:

```
Profile = start; count; indent
```

*Start* is the number of the line in the component where you want the profile indent to begin; *count* is the number of lines for which you want the indent to be in effect; *indent* is the number in inches you want the text to be indented. For example:

```
Left Profile = 1; 5; 2 inches
```

This specification starts a left indent at line 1 of the paragraph component, is in effect for 5 lines, and indents the text 2 inches from the left margin.

You can also specify *Profile = None* for an instance when there is a profile specification on the master but not on the instance.

---

### Note

Make sure the left and right profiles do not overlap so as to exclude the middle space, leaving no room for text. For example, if a left profile specifies an indent of 7 inches and a right profile specifies an indent of 4 inches, there is no room for text since the left and right indents overlap.

The profile properties appear on the Profile sheet of the Component Properties dialog box.

## Contents

The *Contents* property determines prefix and shared content in the component. If the value is set to Private, the master contributes no prefix or shared content to the component instance. It may contribute initial content, but that content can be varied without affecting other instances.

If the value of *Contents* is set to Prefix, the master contributes a prefix to the component instance. A component instance can have Contents set to Private, in which case the instance has no prefix even though the master has Contents set to Prefix.

If the value of *Contents* is set to Shared, the master contributes shared content to the component instance. A component instance can have *Contents* set to Private, in which case the instance has no shared content even though the master has *Contents* set to Shared.

.....

If the value of *Contents* is set to Prefix Shared, the master contributes both prefix and shared content to the component instance. A component instance can have *Contents* set to Private, in which case the instance has no prefix or shared content even though the master has *Contents* set to Prefix Shared.

In the ASCII markup for components with shared material, there is a `<!Class, ...>` declaration for the component with *Contents* set to Shared. The shared-content material follows the *Contents = Shared* specification.

For example, you create a shared-content component called *Greetings*. It has for shared contents the line: *We are pleased to inform you*. The markup for this component is

```
<!Class, Greetings,
    Top Margin = 0.04 inches,
    Bottom Margin = 0.04 inches,
    Line Spacing = 1.162 lines,
    Font = F4,
    Left Tab = 0/1*3 inches,
    Composition = Optimum,
    Contents = Shared>
We are pleased to inform you
```

The ASCII markup for components with prefix material is more complex. First, there is a `<!Class, ...>` declaration for the component with *Contents* set to Prefix. Then there is a `<!Class, ...>` declaration for the prefix with *Contents* set to Shared, followed by the content of the prefix. Finally, there is a specification for the prefix with *Contents* set to Shared. A prefix is indicated by the vertical bar character (`|`), which is followed by a full colon (`:`), followed by the name of the component. That is,

```
|:component name
```

For example, you create a prefix component called *bullet*. The markup for this prefix component is

```
<!Class, bullet,
    Top Margin = 0.04 inches,
    Bottom Margin = 0.04 inches,
    Left Margin = 0.75 inches,
    Right Margin = 0.75 inches,
    First Indent = -0.25 inches,
    Line Spacing = 1.162 lines,
    Font = F4,
    Left Tab = 0/0.50/1//2/3 Inches,
    Composition = Optimum,
    Contents = Prefix>

<!Class, |:bullet,
    Top Margin = 0.04 inches,
    Bottom Margin = 0.04 inches,
    Line Spacing = 1.162 lines,
    Font = F4@i*,
    Left Tab = 0/0.50*3 Inches,
    Composition = Optimum,
    Contents = Shared>

<F6>S<F0><Tab>

<|:bullet,
    Font = @i*,
    Subcomponent = Yes,
    Contents = Shared><F6>S<F0><Tab><End Sub><F0>
```

---

**Note**

The *@i\** convention states that a font used in the prefix will inherit its definition from its surrounding component. For a fuller description of this convention and others related to inherited properties, see *Inline Components* in this chapter.

You can combine prefix and shared content in one component definition. The following example gives the markup for a component called *Prefix\_Shared*, a component with both prefix and shared material.

```
<!Class, Prefix_Shared,
    Top Margin = 0.04 inches,
    Bottom Margin = 0.04 inches,
    Line Spacing = 1.162 lines,
    Font = F4,
    Left Tab = 0/1*3 Inches,
    Composition = Optimum,
    Contents = Prefix Shared>

<!Class, |:Prefix_Shared,
    Top Margin = 0.04 inches,
    Bottom Margin = 0.04 inches,
    Line Spacing = 1.162 lines,
    Font = F4,
    Left Tab = 0/1*3 Inches,
    Composition = Optimum,
    Contents = Shared>
    Prefix for this component

<|:Prefix_Shared,
    Font = @i*,
    Subcomponent = Yes,
    Contents = Shared><F0>Prefix for
        this component<End Sub>
<F0>Shared Content for this component
```

**<!Master Frame, ...> Declaration**

The markup for the *<!Master Frame, ...>* declaration is as follows:

```

<!Master Frame,
    Name = name,
    Placement = At Anchor/Following Anchor/Top/Following
               Text/Bottom/Overlay/Underlay,
    Width = x inches/Column/Page Without Margins/
           Page With Both Margins/Page With Left Margin/
           Page With Right Margin/Page With Outer Margin/
           Page With Inner Margin/Left Page Margin/
           Right Page Margin/Outer Page Margin/Inner Page
           Margin/Gutter/Contents, [2 inches]
    Height = x inches/Page Without Margins/
           Page With Both Margins/
           Page With Top Margin/Page With Bottom Margin/
           Top Page Margin/Bottom Page Margin/Contents,
           [1 inch]
    Horizontal Reference = Page With Both Margins
                        /Page Without Margins/
                        Left Page Margin/Right Page Margin/
                        Inner Page Margin/Outer Page Margin/Column/
                        Left Gutter/Right Gutter/Anchor,
    Vertical Reference = Page With Both Margins/
                       Page Without Margins/
                       Top Page Margin/Bottom Page Margin/Anchor,
    Horizontal Alignment = Right/Left/Center/Inner/Outer,
    Vertical Alignment = Top/Center/Bottom/Baseline/
                       +/- n inches,
    Repeating = Yes/No/Begin/End,
    On Anchor Page = Yes/No,
    Begin on Anchor Page = Yes/No,
    End on Anchor Page = Yes/No,
    Auto Edit = Yes/No,
    Size Contents to Width = Yes/No,
    Size Contents to Height = Yes/No,
    Shared Contents = Yes/No,
    Same Page = Yes/No,
    No Border = Yes/No,
    Not Selectable = Yes/No
    Overlap = Yes/No, (for At-Anchor frames only)
    Numbered= '<Autonum, streamname, level, ...>',
    Superscript= Yes/No,
    Diagram =
...>

```

For information about the markup that follows *Diagram =*, refer to the chapter *ASCII Format for Graphics Objects*.

Starting with Release 5.2, numerical values for both *Width* and *Height* are output automatically, in addition to the frame dimension specifications, if any.

Just as component classes exist apart from any particular component, frame masters exist apart from any particular frame in a document. The representation of frames and frame masters in ASCII format differs from the representation of components and component masters: each master frame and each frame instance is represented by the full complement of properties, not just by the properties that differ from the defaults.

---

**Note**

If an externally created ASCII document contains no *<!Master Frame, ...>* declarations, six default master frames are loaded with the document when it is opened on the desktop. If there are any *<!Master Frame, ...>* declarations in an external document, only these masters are loaded with the document when it is opened on the desktop.

**Name**

If *name* contains any spaces, it must have quotation marks around it (see *Quoting Conventions* in the chapter *ASCII Format Basics*).

**Placement**

If At Anchor is used, the frame is placed at its anchor. The anchor is not visible.

If Following Anchor is used, the frame is located immediately below the text line containing the anchor.

If Top is used, placement is at the top of the page.

When Following Text is specified, placement of the frame is immediately below all text on that page but not necessarily at the bottom of the page. If a Following Text frame does not fit between the anchor and the bottom of the page, the frame is placed at the top of the next page.

Bottom placement of a frame means the frame occurs after its anchor, usually at the bottom of the same page. If there is not room for the anchor and the frame on the same page, the frame is placed at the bottom of the next page, or at the top of the next page if the next page is otherwise empty.



A frame with Overlay placement overlays the other content of the page like a transparent layer. A frame with Underlay placement underlays the other content of the page. Overlay and Underlay frames do not displace text on the page.

## Width

You can have fixed frame width, or you can express the width in terms of page or frame content dimensions. For example, the width can be half the left margin, or column width plus a pica, or equal to the content's width. The generic form of the expression is

$$\text{Width} = (\text{value} * \text{fraction}) + \text{adjustment}$$

The possible values for *Width* are Column, Page Without Margins (Page is equivalent), Page With Both Margins, Page With Left Margin, Page With Right Margin, Page With Outer Margin, Page With Inner Margin, Left Page Margin, Right Page Margin, Outer Page Margin, Inner Page Margin, Gutter, and Contents.

*Fraction* is limited to a range of 0 to 4, and it is omitted if it is equal to one. *Adjustment* may be omitted if equal to 0.

$$\text{Width} = (\text{Page Without Margins} * 0.65) + 0.20 \text{ inches}$$

## Height

The *Frame Height* property is similar to the *Frame Width* property. The values for *Frame Height* are Page Without Margins (Page is equivalent), Page With Both Margins, Page With Top Margin, Page With Bottom Margin, Top Page Margin, Bottom Page Margin, and Contents.

---

### Note

Starting with Release 5.2, the actual values for frame height and width are always written to output regardless of the type of frame specified. Thus, even if a value such as *Contents* is used, the actual width is written to output. By default, these values are in inches.

## Horizontal Reference

Frame *Horizontal Reference* is the page container to which the frame's horizontal alignment refers, and it is only applicable to Overlay, Underlay, and Side frames. For At Anchor frames, this field is meaningless. For other displacing frames, the reference is implicitly *Column*, and you need not specify it. Displacing frames take up room in the text; Overlay, Underlay, and Side frames do not.

For example, if an Overlay frame is right-aligned in the left margin, you specify *Horizontal Reference = Page Without Margins*.

The complete list of horizontal reference values is Page With Both Margins, Page Without Margins, Left Page Margin, Right Page Margin, Inner Page Margin, Outer Page Margin, Column, Left Gutter, Right Gutter, and Anchor.

Anchor is a special setting since it does not define a page container. Left-aligned to anchor means the left edge of the frame aligns with the anchor.

## **Vertical Reference**

The frame *Vertical Reference* property applies only to Overlay and Underlay frames. For At Anchor frames, the reference is implicitly Anchor, and for other displacing frames (Top, Following Anchor, etc.) this field is meaningless.

An example of vertical reference is an Overlay frame bottom-aligned within the top margin specified as *Vertical Reference = Top Page Margin*.

The complete list of vertical reference values is Page With Both Margins, Page Without Margins, Top Page Margin, Bottom Page Margin, and Anchor.

## **Horizontal Alignment**

You can align the frame to the Right, Left, Center, Inner, or Outer edges of the horizontal reference container. The Inner/Outer setting is interpreted for single-sided pages as Right/Left.

You can also specify an offset to that alignment. For left, right, and center alignments, positive offset values mean an offset to the right and negative offset values mean an offset to the left. For inner and outer alignments, positive values mean an offset away from the binding and negative values mean an offset toward the binding.

The generic format for specifying horizontal alignment is as follows:

Horizontal Alignment = *horizontal alignment value + or - offset*

For example, you specify the *Horizontal Alignment* property for an Overlay frame aligned to the left edge of the page and offset 0.10 inch to the right as *Horizontal Alignment = Left + 0.10 inches*.

**Vertical Alignment** The *Vertical Alignment* property applies only to fixed frames, Overlay frames, Underlay frames, Side frames, and At Anchor frames. For floating displacing frames (Top, Following Anchor, etc.), the *Vertical Alignment* property has no effect.

Within the vertical reference container, you can align the frame to the Top, Center, or Bottom edges. You can also align to the Baseline frames that have vertically anchor-relative placement (At Anchor and Overlay frames with a setting of *Vertical Reference = Anchor*). You can also specify an offset. A negative offset value means up; a positive offset value means down.

For example, if a frame is 0.25 inch below center, the specification is *Vertical Alignment = Center + 0.25 inches*.

**Repeating Frame** Repeating frames appear in the same position on consecutive pages. Repeating frames can appear on a series of pages, but not on a series of columns.

A frame can be repeating if it is a shared-content frame with placement of Top, Bottom, Following Text, Overlay, or Underlay.

You specify whether the frame begins or ends a repeat sequence, and whether or not the frame takes effect on its anchor's page. You can use Begin, Start, or Yes to specify a repeating frame that begins a sequence; for example, *Repeating = Start*. You can use End or Finish to specify a repeating frame that terminates a sequence; for example, *Repeating = End*.

To specify that the repeating frame take effect on its anchor's page, use one of the following: *On Anchor Page = Yes*, *Begin on Anchor Page = Yes*, *End on Anchor Page = Yes*.

**Editor** You can associate a frame with object-specific editors—the microdocument text editor rather than the default diagramming editor. The ASCII markup for specifying object editor is *Auto Edit = Yes*.

## Content Auto-Sizing

You use the *Size Contents To Width* and *Size Contents To Height* properties to force the frame contents to fit the frame as closely as possible.

---

### Note

*Frame Width From Contents* and *Size Contents To Width* are mutually exclusive properties; you can turn on only one of these properties for a frame. Similarly, you cannot use *Frame Height From Contents* and *Size Contents To Height* simultaneously on a frame.

## Shared Contents

If you set *Shared Contents* to Yes in the command for a frame instance, the command must not contain markup for the graphic, but there should be graphics markup in the declaration for the master frame.

## Same Page

In ASCII markup, *Same Page* is synonymous with *Same Column*; both appear as *Same Column* on the Frame Properties dialog box. This property specifies whether Interleaf 7 should try to place the frame on the same page (in a single-column document) or in the same column (in a multicolumn document) as the frame anchor.

If *Same Page* is set to Yes, Interleaf 7 places the frame on the same page or in the same column as the frame anchor. If *Same Page* is set to No, the software tries to place the frame on the same page or column as the frame anchor, but no special adjustments are made to accommodate this placement.

## Overlap

*Overlap* may be set to Yes only if *At Anchor* is specified. With *Overlap* selected, a frame and its content can overlap other content on a document page.

## No Border

When *No Border* is set to Yes, the gray marking boundary of an open frame does not appear in a document opened by Interleaf 7.

## Not Selectable

When the *Not Selectable* property is set to Yes, a frame may no longer be selected in Interleaf 7.

## Numbered

The *Numbered* property is for numbered frames. The *Numbered* property markup is the same as the general autonumber property markup. The generic markup for the *Numbered* property is:

```
Numbered = '<Autonum, name, level, [other autonum  
markup]>'
```

An example of markup for the *Numbered* property is:

```
Numbered = '<Autonum, footnote, 1, Tagname = EiXYY2e4vr>'
```

The single quotes surrounding the autonumber markup and the angle brackets are optional. They make for better error recovery.

## Superscript

The *Superscript* property is for numbered frames. The markup has Yes and No as its only values. This *Superscript* property corresponds to the *Superscript* property on the Custom sheet of the Frame Properties dialog box for numbered frames.

## Property Conflicts

Some frame properties are incompatible with others. The Frame Properties dialog box shows how these features interact. For example, the *Repeating* choices only appear if the frame is not anchor-relative and has shared content; you cannot specify *Repeating = Yes* for At Anchor frames with no shared content.

The following lists some of the incompatibilities in ASCII markup for frames:

- A shared frame cannot have *Size Contents To Width = Yes* or *Size Contents To Height = Yes*.
- A frame that has no shared contents cannot have *Repeating = Yes*.
- An anchor-relative frame (*Placement* or *Reference* refers to the anchor) cannot have *Repeating = Yes*.
- A fixed At Anchor frame cannot have the *Numbered* property specified.
- A repeating frame cannot have the *Numbered* property specified.
- *Frame Width = Contents* is incompatible with *Size Contents To Width = Yes*.
- *Frame Height = Contents* is incompatible with *Size Contents to Height = Yes*.
- Baseline alignment is meaningful only with vertically anchor-relative frames.

.....

### **<!Master Diagramming Object,...> Declaration**

For information about the *<!Master Diagramming Object, ...>* declaration, see the chapter *ASCII Format for Graphics Objects*.

### **<!Master Table,...> Declaration**

For information about the *<!Master Table, ...>* declaration, see the chapter *ASCII Format for Tables*.

### **<Master Row... > Declaration**

For information about the *<Master Row, ...>* declaration, see the chapter *ASCII Format for Tables*.

### **<!Comment, ...> Declaration**

You can insert comments in ASCII markup. These comments do not appear when you display the document in Interleaf 7; the ASCII loader eliminates them. This means that if you resave the document in ASCII format, the comment is lost.

The loader ignores everything between *<!Comment, ...* and the closing angle bracket (*>*) in the comment declaration.

If you have embedded matching angle brackets (*<...>*) within the comment, the loader treats them as part of the comment.

### **<!End Declarations> Declaration**

This declaration's function is to mark the conclusion of the series of declarations in an Interleaf ASCII file. It has no effect on input. It is optional but is always generated when a document is saved with Interleaf 7. Its main purpose is to make it easy for filters that want to skip over all declarations.

---

## Commands

### <Autonum, ...> Command — Autonumbers

The markup for saving the current value of an autonumber is:

```
<Autonum, streamname, level, {other autonum markup},
    Value = CurrentValue>
```

For example:

```
<Autonum, list,1,First = Yes,Tagname = vivian,Value = 1>
```

The *<Autonum, ...>* command places an autonumber token in the text. The basic form of the autonumber command is *<Autonum, name, n>*. *Name* is the name of the autonumber stream, and *n* is a number between 1 and 20 indicating the level of the autonumber stream in the command.

There are other properties that can follow the level number. These properties are the *First* property, the *Restart* property, and the *Tagname* property.

If the autonumber token is the first in the stream, the command is, for example, *<Autonum, figure, 1, First = Yes>*.

To restart a numbering stream, use the command *<Autonum, figure, 1, Restart = Yes>*. You must include the *Restart = Yes* property. You can use this property only in the command for a Level 1 autonumber token.

If you plan to make autoreference to the autonumber, the command can include the *Tagname* property. There are two forms of this property:

- When you name the tag, the command looks like this:  
*<Autonum, figure, 1, Tagname = string>*.
- If you let Interleaf 7 name the tag, the command looks like this:  
*<Autonum, figure, 1, Tagname = eIb5W3user>*.

The value of the *Tag* property in the autoreference must be the same as the value of the *Tagname* for the autonumber that is being used as the reference point. An autoreference can refer either to an autonumber or to the page on which the autonumber appears. The form is either *<Ref, Auto #, Tag = Tagname (from Autonum entry)>* or *<Ref, Page #, Tag = Tagname (from Autonum entry)>*.

You can use *Tag* and *Tagname* interchangeably in markup. When you save documents, however, you see *Tagname* in autonumber markup and *Tag* in autoreference markup.

Spaces in an autonumber or autoreference tag must be enclosed in quotation marks. Otherwise, they are stripped out when the document is loaded by Interleaf 7. For more information, see *Quoting Conventions* in the chapter *ASCII Format Basics*.

### **Saving the Current Value of an Autonumber**

When you save a document, you can save the current value of autonumbers. The current value is used when the document containing the autonumber (or the document into which you paste the autonumber) has numbering streams frozen.

To save the current value of an autonumber token, you specify this current value for the *Value* property. The string for the current value string can be no longer than 120 characters.

### **<Ref, ...> Command — Reference Tokens**

The *<Ref, ...>* command creates one of several types of reference tokens. The basic form of the autoreference command is *<Ref, Auto #, Tag = tagname>* or *<Ref, Page #, Tag = tagname>*, where *tagname* is from an *Autonum* entry. The first command causes the reference to refer to the value of the autonumber token, and the second causes it to refer to the number of the page on which the autonumber token appears. *Tagname* is limited to 15 characters. Refer to *<Autonumber, ...> Command* in this chapter for details.

### **Saving the Current Value of an Autoreference**

You can save the current value of an autoreference when you save the document. The current value is used when the document containing the autoreference (or the document into which you are pasting the autoreference) has frozen number streams.



You save the current value by specifying the current value for the *Page Value* property; for example, `<Ref, Auto #, Page Value = 1>`. The value string can be no longer than 120 characters.

### Attribute Reference

An attribute reference displays the value of a user-defined attribute over a specified range. The range can be a component, inline, page, or sheet (front and back of a page). The attribute reference can display the minimum or maximum of the values over that range by setting *Operation* = *Min* or *Max*.

The following is an example of attribute reference markup:

```
<Ref Attribute Value, Range = Sheet, Operation = Max,
  Attribute = "string (name of the attribute)",
  Result = "attribute value",
  Lookup Pairs = "NIL" "0" "zero" "1" "one" "2" "two">
```

### Show Blank Pages Reference

The *Show Blank Pages* reference indicates that the following page is blank. For example, a page at the end of a chapter could be blank. A reference on the facing page can indicate that the following page is blank.

The following is an example of *Show Blank Pages* reference markup:

```
<Ref, Show Blank Page, Page Value = a>
```

*Show Blank Pages* is primarily useful for double-sided documents. For single-sided documents *Show Blank Pages* only has an effect when the document is in a book.

### Table Page Number Reference

Use the *Current Frame* and *Total Frame* reference properties to number multi-page tables; for example, *Sheet 2 of 10*. You determine the start of the range with the *Starting Num* property.

The following are examples of this type of reference markup:

```
<Ref, Current Frame>
<Ref, Total Frames, Starting Num = 6>
```

## <component, ...> Command

The markup for the <component, ...> command is identical to that for the <!/Class, ...> declaration with the following exceptions (markup common to both <!/Class,...> and <component, ...> has been omitted, see <!/Class...>):

```
<component ,
    @attribute = value, (see User-Defined Attributes in this
        chapter)
    Hidden = Yes/No,
    Read Only = Yes/No,
    Font = Fn,
    Fill = Yes/No/On/Off/Blank
    A-Page Added Hyph = Yes/No,
    A-Page Added FJ = Yes/No,
    A-Page Removed HCR = Yes/No,
    A-Page Widow = n,
    A-Page Glues = n,
    A-Page Bottom Margin = x inches,
    Subcomponent = Yes/No>
```

### Name

The *component* in the <component, ...> command is the name of the component; for example, <para, ...>. In the body of the document, the command <para> indicates the beginning of a *para* component. If the properties of a component are identical to the properties of the component master (<!/Class, para, ...>), you need specify only the name of the component, <para>.

If the properties are not identical to those of the component class, you must specify the properties that are different within the command. For example, <para, Font = F3> indicates that the font of this paragraph is different from the font in the class declaration for the *para* component.

### Fill

The component property *Fill = off* inserts a hard return at the end of each line. This is useful for components with tabular material.

When *Fill* = *blank*, the loader inserts a blank at the end of each input line if no blank was there and if it is not the last line of a component. This is useful for documents prepared with word processors or editors that do not ensure blanks at the ends of lines. When line breaks on the word processor differ from those in Interleaf 7, *Fill* = *blank* keeps words from running together. Usually, it is easiest to put this command in a `<!Class Defaults, ...>` declaration so that it applies to all component classes.

## Hidden = Yes/No

If a document has a hidden component (or frame, or table row) when you save the document, the ASCII dumper program outputs *Hidden* = *Yes* as a property for that document element.

Output filter programs can search for *Hidden* = *Yes* to eliminate all hidden document elements. *Hidden* = *Yes* generally appears in markup to indicate that an element of a document is hidden by a control expression.

However, *Hidden* = *Yes* may also indicate that an element is part of a master's content, for example, a component that is part of the contents of a master frame, a frame that is part of the content of a master component, and a component in a frame that is part of a master component. Since table cells are part of the content of table row masters, the phrase *Hidden* = *Yes* appears in the ASCII markup for any Master Row definition. Master Row definitions appear in the markup for both table masters and table instances. For example:

```
<!Master Table, "3x3",
    Columns =      3,
    Column 1 Width = 1 units,
    Column 2 Width = 1 units,
    Column 3 Width = 1 units,
    Row =          "row",
    Row =          "row",
    Row =          "row">
<Master Row, "row">
<Cell><"3x3:cell",
    Hidden =       yes>
<Cell><"3x3:cell",
    Hidden =       yes>
<Cell><"3x3:cell",
    Hidden =       yes>
<End Table>
```

.....

Similar markup appears for each Master Row definition present in markup for a table instance. Use of this phrase to indicate that an element is part of a master is not limited to table markup, but always occurs in table markup.

---

**Note**

Both *Hidden = Yes* and *Hidden = No* are ignored by the Interleaf ASCII loader.

**Read Only  
=Yes/No**

Allows you to specify any component as a read-only component. Read-only components cannot be edited in Interleaf.

**A-Page**

When you use the Misc→ A-page→ Split command to split a component, inline, or table between two or more documents, the ASCII dumper writes out ASCII A-page properties for the split component, inline, or table. These ASCII properties contain information about the state of the document when it was split. Interleaf 7 uses this information if the split document is joined using the Misc→ A-page→ Join command.

Do not attempt to modify these properties. Changing their values or trying to generate them initially in ASCII format can have uncertain results and cause your document to rejoin incorrectly.

In general, Interleaf 7 divides documents at existing page breaks. If a component, inline component, or table straddles a page break, Interleaf 7 splits the component, inline, or table between the two pages. The A-page ASCII properties occur for the instance of the component, inline, or table that was split. These properties occur only for the split instance, never for a master component, inline, or table.

The ASCII A-page properties are:

- A-Page Added Hyph = Yes/No, whether Interleaf 7 added a hyphen to the word where the A-page split occurred.
- A-Page Added FJ = Yes/No, whether Interleaf 7 added a force-justify token where the A-page split occurred.
- A-Page Removed HCR = Yes/No, whether Interleaf 7 removed a hard carriage return where the A-page split occurred.
- A-Page Widow = *n*, the setting for widow control at the point where the A-page split occurred.

- A-Page Glues =  $n$ , how many glues (soft spaces) were removed at the point where the A-page split occurred.
- A-Page Bottom Margin =  $x$  inches, the setting for the bottom margin at the point where the A-page split occurred.

These properties and their values are written in the component definition for the start of the component now split. They are generated automatically when needed, so you need not touch them. Modifying them can cause your document to fail to rejoin or to rejoin incorrectly.

## Inline Components

You specify inline components by setting the property *Subcomponent* = *Yes*. Only a subset of component properties apply to inline components. These are:

Font family, size, and typeface  
 Color  
 Underline  
 super/subscripts  
 strikethrough  
 overbar  
 caps  
 rev bars  
 dictionary  
 pair kerning  
 track kerning  
 Placement: begin new page, begin new column  
 Allow breaks: within inline yes/no  
 Content  
 Table of Contents

There are no format options associated with margins, no tabs, and no hyphenation available for inline components. Specifying any of these properties has no effect.

The contents of the inline component follow the component command. The end of the inline component is marked with the token *<End Sub>* or *<End Inline>*. The ASCII loader accepts either *<End Sub>* or *<End Inline>* but only produces *<End Sub>*.

.....

A font token must appear first in the contents of the inline component, and a font token must appear immediately after the *<End Sub>* marker.

Typical markup for an inline component is

```
<name,  
  Font =  
  Subcomponent = Yes> <Fn>  
  contents of inline component <End Sub><F0>
```

An example of markup for an inline component is as follows:

```
<Prefix:bullet,  
  Font = @i*,  
  Subcomponent = Yes,  
  Contents = Shared><F6>S<F0><Tab><End Sub><F0>
```

An inline component can only occur inside a component or inside another inline component. For information on font inheritance properties for inline components, see *Inline Component Inheritance* in the section *Inline Components* in this chapter.

*Inline*, *Inline Component*, *Subcmpn*, and *Subcomponent* are synonymous in the ASCII loader. *End Sub* and *End Inline* are synonymous in the ASCII loader.

## Inline Component Inheritance

You can specify inheritance for inline component default font properties. Font inheritance applies to inline components only, but must also be expressed for any master component that acts as a master for inline components that use inheritance.

You can specify inheritance for each text property, as well as for font properties such as Family, Size, Bold, and Italic. In addition to setting a property to inherit, you can set relative inheritance for many properties; for example, inherit and toggle, inherit and increase, or inherit and decrease.

To specify a text property for the default font of an inline component or master, the syntax is

```
Font = Fn @property{value},
```

*@property* is the property-specifier character for that property; for example, U for underscore, S for strikethrough. {*value*} is property-specific information; for example, a track kerning value. The property must be preceded by the @ symbol.

---

**Note**

Text property inheritance markup cannot be used in <Fn>font tokens.

The syntax for specifying that an inline component inherit a text property from its surrounding component is

```
@iproperty
```

The syntax for specifying that a text property be inherited with some operation is

```
@iop attr
```

The possible values of *op* are: + (inherit and increase), - (inherit and decrease), ~ (inherit and toggle).

The syntax you use to specify that the inline component inherit all text properties is

```
@i*
```

The syntax for specifying that a text property's value is off (or on, in the case of pair kerning) is

```
@~attr-
```

In this situation, the ~ character functions as a negation operator; when it is used in specifying inheritance, it expresses a toggle relationship.

You can specify font properties separately using the following syntax: @F for family, @H for size, @E for bold, @I for italic.

Each succeeding specification can override previous specifications. If an *F<sub>n</sub>* entry is the first specification, all information inherent in that font number is assumed. If an *F<sub>n</sub>* entry occurs farther along in the specification, only the family information pertains. Anything not specified is given the default value.

## Examples

The following examples show how to use font inheritance syntax for inline components:

- To specify an inline component that has default font F4 and inherits nothing:

```
Font = F4
```

- To specify an inline component that inherits all font properties:

```
Font = @i*
```

- To specify an inline component that inherits all font properties, but toggles the italic setting:

```
Font = @i* @i~I
```

- To specify an inline component that inherits all font properties, but is always underlined:

```
Font = @i*@U
```

- To specify an inline component that has default font F4, inherits underlining, and is always all uppercase:

```
Font = F4 @iU @C
```

- To specify an inline component that inherits all properties, but is never bold:

```
Font = @i* @~E
```

Going in order through the six previous examples, with F0 as an appropriate default font, the default fonts assigned to the master are: F68; F0; F0; F0@U; F4@C; F0, unless F0 was bold, in which case it is a font identical to F0 with bold off.



You must specify inheritance of fonts only for the default font of master components and inline components. It must never appear within the contents of the component or as the default font for a regular component.

You can specify inheritance for master components so that they act as masters for inline components. Since they are also masters for regular components, you can specify a default font for component masters. For master components, if there are any inheritance characteristics specified, the default font value assigned is the font specified by all the other information. If only inheritance information is specified, Interleaf 7 assigns an appropriate default value.

### **<Table, ...>, <Row, ...>, and <Cell, ...> Commands**

For information about the <Table, ...>, <Row, ...>, <Cell, ...> commands, refer to the chapter *ASCII Format for Tables*.

### **<Fn> Command — Font Change**

The <Fn> command specifies a font change. It specifies a type family, a size, and a choice of bold, italic, or neither.

The numbered fonts are defined in the <!Font Definitions, ...> declarations that appear at the beginning of the document. The Interleaf font code numbers used when a document is saved change from one release of the software to the next, and they may change if new fonts are installed or if the file is saved in ASCII format while zoomed. You can use numbers other than the Interleaf font code numbers. When you save the document in ASCII format, Interleaf 7 converts your font numbers to the code numbers Interleaf 7 uses, as long as you have used the font numbers consistently in the document.

An example of the markup for a font change is

```
<F6>text in the new font <F0>
```

<F6> specifies the font for the words following the command, and <F0> specifies a return to the default font for the component.

You can also use a non-default font number in the command at the end of the font change.

.....

## Text Properties

You can specify the following text properties in ASCII markup:

- @U = underscore
- @S = strikethrough
- @T = superscript
- @B = subscript
- @R = revision bars
- @P = pair kerning off
- @K = track kern
- @O = overbar
- @D = double underbar
- @Z = color number
- @L = language for spelling/hyphenation dictionary
- @A = all small caps
- @C = all caps
- @X = caps and small caps

---

**Note**

@Z0 is white text. See the *<!Color definitions...>* declaration.

The markup for these text properties must occur immediately after the font number and must be preceded by the @ symbol. For example, *<F6@U@S>* indicates the text that follows is Times 10-point italic with underlining and strikethrough

---

**Note**

The markup F0 cannot be used with @R.

You specify the *language* text property with @L plus a two-letter language code. The default is American English. To specify Norwegian as your spelling hyphenation choice, the markup is the following:

*<F4@Lno>*

The language codes are as follows:

American English	<i>am</i> (default)
British English	<i>br</i>
Canadian French	<i>ca</i>
Danish	<i>da</i>
Dutch	<i>du</i>
Finnish	<i>fi</i>
French	<i>fr</i>
German	<i>ge</i>
Italian	<i>it</i>
Norwegian	<i>no</i>
Portuguese	<i>po</i>
Spanish	<i>sp</i>
Swedish	<i>sw</i>
No language	<i>nl</i>

The loader loads documents containing no markup with the default of American English.

Long ASCII documents that do not need to be hyphenated load much faster with a language of *No Dictionary*.

## Kerning

You can specify kerning properties at the component level or at the text level. You cannot specify these kerning properties for a component master at the *<!/Class,... >* level.

The following track kerning property values correspond to the menu selections in Interleaf 7:

Menu	Value Meaning	Kerning Markup Value
T1	tight	K1
T2	tighter	K2
T3	tightest	K3
T4	loose	K4
T5	looser	K5
T6	loosest	K6

## Kerning as a Text Property of the Font

Pair kerning is on by default, so you do not have to specify it. You specify *Pairs off* with the letter *P*.

Kerning is a text property of the font. An @ always precedes the kerning property.

Some examples of specifying kerning properties on instances of a *para* component are:

```
normal (track off, pair on) <para, ..., Font = F5, ...
pair kerning off, track off <para, ..., Font = F5@P, ...
T1                          <para, ..., Font = F5@K1, ...
T2                          <para, ..., Font = F5@K2, ...
T3                          <para, ..., Font = F5@K3, ...
L1                          <para, ..., Font = F5@K4, ...
L2                          <para, ..., Font = F5@K5, ...
L3                          <para, ..., Font = F5@K6, ...
```

Some examples of kerning changes for a range of text are:

```
pair kerning off           <F7@P>
tight track kerning        <F7@K1>
tighter track kerning      <F7@K2>
tightest track kerning     <F7@K3>
loose track kerning        <F7@K4>
looser track kerning       <F7@K5>
loosest track kerning      <F7@K6>
```

---

### Note

Markup for text properties cannot be added to a default *<F0>* font token.

## </F> Command — Previous Font

The *</F>* command causes the font to revert to that used previously in the component. It is not possible to revert to a font in another component.

You use this command only on input; when you save a document, Interleaf 7 converts the *</F>* command to a numbered font.

Sixteen levels of fonts are allowed within a component. The stack of fonts is reset at the beginning of each component.

**<FI> Command — Italic**

The *<FI>* command specifies a font change to italic in the current type family, if an italic font exists for that family. You terminate this command with *</F>*.

You do not have to declare the resulting font in the *<!Font Definitions, ...>* declaration.

**<FB> Command — Bold**

The *<FB>* command specifies a font change to bold in the current type family, if a bold font exists for that family. You terminate this command with *</F>*.

You do not have to declare the resulting font in the *<!Font Definitions, ...>* declaration.

**<FBI> Command — Bold Italic**

This command specifies a font change to bold italic in the current type family, if a bold-italic font exists for that family. You terminate this command with *</F>*.

You do not have to declare the resulting font in the *<!Font Definitions, ...>* declaration.

**<FJ> Command — Force-Justify**

The *<FJ>* command specifies a force-justify return. You use it at the end of a line to force the text before it to fill the entire line width. It has the properties of a hard return unless the component has justified alignment.

## <Frame, ...> Command

The markup for the <Frame, ...> command is identical to that for the <!/Master Frame, ...> declaration with the following exceptions (markup common to both <Frame, ...> and <!/Master Frame, ...> has been omitted).

```
<Frame,
    @attribute=value, (see User-Defined Attributes in this chapter)
    Hidden = Yes, (see User-Defined Attributes in this chapter)
    Page # = n,
    Anchor Visible = Yes/No
    Force Hidden = Yes/No
    Overlap = Yes/No
    Not Selectable = Yes/No
    No Border = Yes/No
    ...
    ...>
```

The <Frame, ...> command repeats the full complement of properties declared under <!/Master Frame, ...>, not just those that differ from the master frame declaration. Also, defaults are taken not from the master, but from the defaults listed in Appendix A, *Default Values for ASCII Format Documents*.

If *Contents = Shared* is specified in the <Frame, ...> command, do not include markup for the diagram in the command. Put this markup in the declaration for the master frame with the same name.

In the markup for all frames except header/footer and *At Anchor* frames, there is the entry *Page # = n*, where *n* is the number of the page, including any prefix that is present, that contains the anchor.

Page number values will appear on output but have no effect on input.

### Hidden = Yes/No

If a document has a hidden frame, table row, or other component when you save, the ASCII dumper outputs *Hidden = Yes* as a property for that document element.

Output filter programs can search for *Hidden = Yes* to eliminate all hidden document elements. Document elements that are part of a master's content can also take a *Hidden = Yes* specification. These elements include a frame that is part of the contents of a master component. If a frame is marked *Hidden = Yes*, there is no *Page# = n* property for the frame.

Both *Hidden = Yes* and *Hidden = No* are ignored by the Interleaf ASCII loader.

<b>Anchor Visible</b>	When <i>Anchor Visible</i> is set to No, the anchor securing a frame does not appear on the text page following input.
<b>Force Hidden</b>	If <i>Force Hidden</i> is specified as Yes, the frame and its contents will not be visible on the text page following input.
<b>Overlap</b>	When <i>Overlap</i> is set to Yes, the frame can be positioned so as to overlay other text.
<b>Not Selectable</b>	When <i>Not Selectable</i> is set to Yes, the frame cannot be selected for editing.
<b>No Border</b>	When <i>No Border</i> is set to Yes, the bounding box of the frame is not displayed.

### **<HR> Command — Hard Return**

The *<HR>* command represents a hard return.

### **<SR> — Soft Return**

*<SR>* is the Interleaf markup designation for a soft return.

Soft returns are ignored by the ASCII loader. They occur in files dumped by the ASCII dumper to indicate the places in the document where lines breaks occur. Line breaks are recomputed when the file is loaded.

### **<!--Include, ...> and <!--Include Declarations, ...> Commands**

Although *<!--Include, ...>* and *<!--Include Declarations, ...>* look like declarations, they are classified as commands. You can use the *<!--Include Declarations, ...>* command at the beginning of a file to incorporate a set of declarations. You can use the *<!--Include, ...>* command throughout a file to incorporate other files into the current file. Refer to the chapter *The Include Commands* for details.

## <Index, ...> Command

The *<Index, ...>* command creates an index token. Index tokens are inserted in text where the index reference begins. You can specify other markup between *Heading* and the closing angle bracket (*>*), including the range of text that the index token covers.

The markup for the *<Index, ...>* command is as follows:

```
<Index, Heading
  Doc = string, (Index Document Name)
  Dictionary = am (two-letter language dictionary code)
  Autonumber Stream = name
  Page Number Stream Name = name
  Style Fun = "Lisp style function"
  Style Name = "Lisp style function name"
  Page# Sep = "Markup to separate page numbers"
  Lisp = Any Lisp on the Top Level Index Heading
  "Headings"
  Sort, Sort String = sort string for that heading
  Lisp = Any Lisp on that heading
  Typeface = bold/italic
    Count = n, (number of components included in range)
    Pages = n
    To Named = component name, (where reference ends)
    See, See Also = entry name,
    To Next = (range to next component),
  Lisp = Any lisp on index token>
```

## Headings

*Heading* is the text of an index entry. There can be spaces in *Heading* entries, but quotation marks are not strictly necessary as they are elsewhere in ASCII markup. Quotation marks are necessary, however, if you used any of the other characters discussed in *Quoting Conventions* in the chapter *ASCII Format Basics*. Use quotation marks to distinguish *Headings* from other markup.

## Range

Range determines the pages associated with an entry; it can also create a cross-reference.

- *This Page* is the default; no further markup is necessary.
- If range is *To Next*, the markup is *To Next*, which means the range of the entry ends with the next component.
- If range is *Count*, the markup must include the number of components to be included in the reference (*Count = n*).



- If range is *To Named*, the markup must include the name of the component at whose next instance the reference ends (*To Named = component name*).
- If the range is *See* or *See Also*, the markup must include the name of the entry to which you are referring.

## Other Index properties

Other properties you can include in an index token command are

- a *Sort* or *Sort String* property to define a customized sort string; when you use *Sort* or *Sort String*, the entry must follow the corresponding *Heading* entry.
- a *Doc* property to designate a special *Index Document* name; this corresponds to the *Index Document* field on the Custom sheet of the Index Properties dialog box.
- an *Index* property if you want the index to have a name other than *index*. This string is accessible from the Create Index dialog box, in the Index Document Properties dialog box .
- a *Typeface* property to designate a different weight or slant for the reference; the default font for index entries is a 10-point roman typeface.
- a *Dictionary* entry to designate the language dictionary for the index; the *xx* value can be any of the two-letter language codes described in *Text Properties* in this chapter. The default is *am*.

## <|, ...>, Page Break Command

The <|, *string*> command indicates a page break. *String* is the page number including any prefix content.

For example, <|,2> denotes page 2 in Arabic numerals without a prefix; <|, "Page 2"> denotes page 2 with a prefix; and <|,ii> denotes page 2 in lowercase roman numerals without prefix content.

In the markup for all frames except header, footer, and At Anchor frames, there is the entry *Page # = n*, where *n* is the page number, including any prefix content. These page numbers appear only when you save a document in ASCII with Interleaf 7. They are ignored when the document is loaded.

The `<|, ...>` page token markup appears immediately before the first character (or frame, tab, or other marker) on each page. If your component starts a page, the `<|, ...>` appears after the component header and before the first character. If a component starts a page and the component starts with a hidden inline component,

`<|, ...>` precedes the hidden inline component. Alternatively, if a hidden inline component sits at a line break, the `<|, ...>` markup follows it unless there is a `<HR>` or `<FJ>` token before the hidden component, in which case the `<|, ...>` follows the `<HR>` or `<FJ>`.

The page break command does not force a page break. It is ignored by the Interleaf ASCII loader. It is useful in filters that interpret Interleaf ASCII markup since it indicates where page breaks occur. The only way to force a page break is with the Component property *Begin New Page = Yes*.

---

**Note**

Do not confuse the `<|, ...>` command with the `<|:>` command. The `<|:>` command indicates a prefix component, not a page break.

## **<Page Header, ...> and <Page Footer, ...> Commands**

If you do not include any header or footer information in your markup, the ASCII loader uses the defaults, and you can add the text you want with Interleaf 7. When using a filter to create an Interleaf document, leaving out header and footer information and entering it interactively is the simplest and most practical way to handle headers and footers.

---

**Note**

A document can have only one page header command (or set of page header commands). This header remains in effect for the entire document; you cannot change it in the middle.

The page header command(s) determine the format for all pages, an optional special first page that is different, or for left and right pages for double sided documents. These commands are:

`<Page Header (Footer), Frame = ...>` indicates that the document is to be printed in single-sided format and that the first page has the same header or footer as the rest of the document.

`<First Page Header (Footer), Frame = ...>` indicates that the first page has a different header or footer from the rest of the document.

`<Right Page Header (Footer), Frame = ...>` indicates that the document is to be printed in one of the double-sided formats and that this is the header or footer for the right-hand pages.

`<Left Page Header (Footer), Frame = ...>` indicates that the document is to be printed in one of the double-sided formats and that this is the header or footer for the left-hand pages.

For details about the markup inside frames, see the chapter *ASCII Format for Graphics Objects*.

### Rotated Headers and Footers for Turned Pages

Once you declare a value for the *Turned Page* property in the `<!Page..>` declaration, you can declare the header and footer rotation as clockwise or counterclockwise. For the *Rotation* property in the `<Page Header>` and `<Page Footer>` commands, you specify 90 for clockwise rotation and 270 for counterclockwise rotation. The value for the *Rotation* property in `<Page Header, ...>` and `<Page Footer, ...>` must match the value that you specified for the *Turned Page* property of the `<!Page..>` declaration.

### <SP> Command — Hard Space

In Interleaf ASCII markup, only the hard space has a special command: `<SP>`.

### Other spaces

Markup for em, en, hairline or thin spaces follows the procedures used for special characters. See *Special Characters* in this chapter and Appendix C, *Hexadecimal Codes*, for more information.

### <Tab ...> Command

Interleaf 7 has five types of tab stops: left, right, center, numeric, and decimal. There are five kinds of tab appearances: blanks, dots, lines, underlines, and dashed tabs.

The tab stops are indicated in the `<!Class, ...>` declarations for each component master. Tab appearances are indicated by commands in text. The tab commands cause the text to move to the next tab stop.

Depending on the tab command used, the tab space is left blank or filled with dots (....), lines (—), underscores (\_\_\_), or dashes (---).

.....

The ASCII format commands that determine tab display are

- `<Tab>` for blanks (---->) (You may see an arrow on the screen to indicate a blank tab, but it will not appear in a printed document.)
- `<Tab.>` for dots ( . . . )
- `<Tab->` for lines (——)
- `<Tab_>` for underlines (\_\_\_\_)
- `<Tab-dash>` for dashed tabs (---)

Tabular material prepared outside Interleaf 7 conventions might require some cleanup. If you are using a fixed-width font (a type family in which each character occupies an equal amount of space on a line) for tables, there might be a difference between the amount of horizontal space you expect the characters to occupy and the space they actually require in Interleaf 7.

The difference might be more noticeable if your tables were created with a fixed-width type family and Interleaf 7 converts them to a proportional type family, in which characters occupy varying amounts of horizontal space.

---

**Note**

The ASCII tab character in a file is treated the same way as `<Tab ...>`.

## Special Characters: the `<#xx>` Command

The `<#xx>` command represents the hexadecimal value of special characters such as vowels with acute, grave, and circumflex accents. The `xx` represents two to four hexadecimal digits. For example, `ã` is represented in Interleaf ASCII markup as `<#e3>`.

The first step in creating markup for a special character is determining the associated hexadecimal code, which consists of a two-digit number or a number and letter. For a list of these codes, refer to Appendix C, *Hexadecimal Codes*.

## User-Defined Attributes

You can specify user-defined attributes and values for various document elements. User-defined attributes are part of the Conditional Content and Attributes features of Interleaf 7. You distinguish user-defined attributes from ordinary properties by preceding the attribute name with the @ symbol.

The generic specification for a user-defined attribute is

```
@attribute = value,
```

For example:

```
@author = joyce,
```

User-defined attributes should appear at the beginning of the list of properties and values for a declaration or command. Individual instances of a component must also be tagged for the attribute.

In Conditional Content control, user-defined attributes can have multiple values. You specify attributes that have multiple values as follows:

```
@attribute = value1,  
@attribute = value2,  
@attribute = value3, ...
```

For example:

```
@platform = Sun,  
@platform = "DEC ULTRIX",  
@platform = "DEC VMS",  
@platform = HP,  
@platform = Apollo,
```

You never specify a user-defined attribute without a value.

User-defined attributes and values can contain spaces and special characters; use the ASCII format quoting conventions.

You can specify user-defined attributes and values for document components, frames, inline components, and table rows. You can also specify user-defined attributes and values for microdocument components and microdocument inline components. Named graphics objects also take attributes and values.

.....

---

**Note**

There is no ASCII format for defining the attribute type as either *string* or *enumerated*. You specify attribute type interactively in Interleaf 7. If you enter a new user-defined attribute in ASCII format, it has the default characteristics of *String Type* and *Max Number of Values = 1*.

In ASCII format, a user-defined attribute specification on the master is not created automatically for the instance. If an instance has user-defined attributes, you must specify them with the instance even though they were specified on the master. If you do not specify a user-defined attribute for the instance, the default is no user-defined attributes.

For example, the `<!Class, ...>` declaration for the *bullet* component has a user-defined attribute in the list of properties and values as follows:

```
<!Class, bullet,
  @annotated = Yes,
  Top Margin = 0.04 inches,
  Bottom Margin = 0.04 inches,
  Left Margin = 0.75 inches,
  Right Margin = 0.75 inches,
  Line Spacing = 1.162 lines,
  Font = F14,
  Left Tab = 0/0.50*3 Inches,
  Composition = Optimum,
  Contents = Prefix>
```

Even though the user-defined attribute, `@annotated = Yes`, is specified on the master in the `<!Class, ...>` declaration for the *bullet* component, you must specify it again in the `<component, ...>` command for the instance of the bullet component as follows:

```
<bullet,
  @annotated = Yes,
  Font = F18>
```

## <Comment ...> Command

You can insert comments into markup with the *<Comment...>* command. These comments do not appear when you display the document in Interleaf 7; the ASCII loader eliminates them. This means that if you resave the document in ASCII format, the comment is lost.

The loader ignores everything between *<Comment, ...* and the closing angle bracket (*>*) in the comment command.

If you have embedded matching angle brackets (*<...>*) within the comment, the loader treats them as part of the comment.

---

**Note**

If a component with no content except for a soft space is saved in ASCII format, *<Comment>* appears in the output file. This insures that should the file be transmitted using a UUCP mailer, the empty component is not interpreted as an end-of-file.

# *ASCII Format for Tables*

# 3

This chapter describes the Interleaf ASCII format for tables. It also describes the properties you can specify for tables.

A table is similar to a series of components in the general structure of a document. It differs from a series of components, however, in that it has a distinct internal structure. A table contains columns, rows, and cells.

Each table has its own set of master rows. Columns, however, do not have masters. Cells grow and shrink to fit their contents. Cells can adjust to align themselves with other cells in the row and column. Columns can have fixed width or proportional width. Proportional-width columns change width to take into account changes to the widths of other columns.



---

## Table Declarations and Commands

### **<!Master Table, ...> Declaration**

The *<!Master Table, ...>* declaration for a table is similar to the *<!Class, ...>* declaration for a component. The *<!Master Table, ...>* declaration specifies the general form and properties of a table.

An instance of a table generally uses the properties of its master as defaults. If the instance has more columns than its master, the additional columns use the column default values. For these and other table default values, see Appendix A, *Default Values for ASCII Format Documents*.

The markup for the `<!--Master Table, ...>` declaration is as follows:

```
<!--Master Table, name,
    @attribute = value,
    A-Page = Yes/No,
    Allow Page Break After = Yes/No,
    Allow Page Break Within = Yes/No,
    Allow Page Break Before = Yes/No,
    Begin New Column = Yes/No,
    Begin New Page = Yes/No,

    Column N Width = x inches,
or
    Column N Width = x units,
or
    Column N Width = x units + x inches,

    Column N Top Ruling Visible = Yes/No,
    Column N Left Ruling Visible = Yes/No,
    Column N Top Ruling Weight = 0.125 to 6.125 [1] Double,
    Column N Left Ruling Weight = 0.125 to 6.125 [1] Double,
    Top Margin = x inches [0.08 inches],
    Bottom Margin = x inches, [0.08 inches],
    Left Margin = x inches [0],
    Row = name,
    Rulings to Bottom = Yes/No
    Table Page Break Rulings = None/Top/Bottom/Both,
    Straddle = Yes/No,
    Orphan Control = 1 - 16 [2 rows],
    Widow Control = 1 - 16 [2 rows],
    Top Border Visible = Yes/No,
    Bottom Border Visible = Yes/No,
    Left Border Visible = Yes/No,
    Right Border Visible = Yes/No,
    Header Border Visible = Yes/No,
    Footer Border Visible = Yes/No,
    Top Border Weight = 0.125 to 6.124 [1] Double,
    Bottom Border Weight = 0.125 to 6.124 [1] Double,
    Left Border Weight = 0.125 to 6.124 [1] Double,
    Right Border Weight = 0.125 to 6.124 [1] Double,
    Header Border Weight = 0.125 to 6.124 [1] Double,
    Footer Border Weight = 0.125 to 6.124 [1] Double
    Top Border color = n,
    Bottom Border color = n,
    Left Border color = n,
    Right Border color = n,
    Header Border color = n,
    Footer Border color = n>
```

<b>Attribute</b>	Tables may now have user-defined attributes.
<b>Table A-Page</b>	The <i>A-Page</i> property is for internal use only. Do not specify it.
<b>Table Allow Page Break</b>	<p>When <i>Allow Page Break Within</i> is set to Yes, Interleaf 7 can put part of the table on one page and part of it on the next page as long as this does not violate the widow/orphan settings for the table. If <i>Allow Page Break Within</i> is set to No, the table will not break across page boundaries. In multicolumn documents, the <i>Allow Page Break Within</i> setting also determines whether a table breaks across columns on the same page.</p> <p>When <i>Allow Page Break Before/After</i> is set to Yes, <b>Interleaf 7</b> determines page breaks. When <i>Allow Page Break Before/After</i> is set to No, the software does not permit a page break between a table and the preceding or following component. In multicolumn documents, the <i>Allow Page Break Before/After</i> setting also determines whether a column break is permitted to occur before or after a table begins.</p>
<b>Table Begin New</b>	If set to Yes, the <i>Begin New Page</i> property forces the table to begin on a new page. You can also force a table in a multicolumn document to begin a new column by setting <i>Begin New Column</i> to Yes. The <i>Begin New Column</i> property does not apply to single-column documents.
<b>Columns</b>	On input, if the <i>&lt;Table, ...&gt;</i> command or <i>&lt;!Master Table, ...&gt;</i> declaration does not have <i>Columns = n</i> , Interleaf 7 derives the number of columns from the master. If there is no master, the number of columns is 1. The software then raises this number to the highest column number specified in any of the <i>Column N</i> properties, such as the column number specified in <i>Column N Width</i> .
<b>Column N Properties (Fixed-Width)</b>	<p>These are the properties for fixed-width columns. The <i>Column N</i> properties refer to a column by a number <i>N</i>. Columns are numbered starting on the left from column number 1.</p> <p><i>Column N Width</i> determines the width of column <i>N</i>. For example, <i>Column 3 Width</i> specifies the width of the third column from the left in the table.</p>

On input, you can use *Column N Top Ruling Visible*, *Column N Left Ruling Visible*, *Column N Top Ruling Weight*, and *Column N Left Ruling Weight* in both `<Table, ...>` and `<!--Master Table, ...>` markup. If a table's column rulings differ from its master's, the individual cells reflect this difference with the *Left Ruling* and *Top Ruling* properties.

When *Column N Ruling Weight* is specified, it implies that *Column N Ruling Visible* is set to Yes. Specifying Double for *Column N Top Ruling Weight*, and *Column N Left Ruling Weight* as optional, makes a double ruling of the specified weight.

If you specify *Column N Width*, the column rulings for that column are reset to the default values. Thus, *Column N Width* must precede every *Column N Ruling Weight* for column *N*.

**Proportional-Width** A proportional-width table allows specified columns to adjust in width if there are any changes made in width by any other column or margin. Adjustment is proportional to the width of the entire table. Columns marked as proportional-width are declared in *units* rather than inches.

**Table Margin** *Top Margin* determines the distance between the table and the component above the table. *Bottom Margin* determines the distance between the table and the component below. *Top Margin* has no effect at the top of a page; *Bottom Margin* has no effect at the bottom of a page. *Left Margin* determines the distance from the left page margin to the left edge of the table.

**Table Row** You use the *Row* property only in master tables. On input, you must list in the master the name of each row in the table. The order of row entries is significant and must reflect the order in which they appear in the master.

On output, there is one *Row* entry for each row you created interactively with the `Create→Table→tablename` command. See the sample table markup for examples of how to specify rows in the master.

**Table Page  
Break Rulings**

The *Table Page Break Rulings* property determines whether table rulings appear at page breaks. *Table Page Break Rulings* are printed at the top of the page if Top is specified and at the bottom of the page if Bottom is specified. With None specified, no rulings are printed; with Both specified, top and bottom rulings appear.

**Rulings to  
Bottom Property**

This property specifies whether table ruling goes to the bottom of the page. This is the military specification (milspec) treatment of tables where the outside rulings fill the page no matter where the rows stop. The default is No.

**Table Straddle  
Property**

The *Straddle* property determines whether a table extends horizontally across the entire text area of a multicolumn document. If Yes is specified, the table spans the entire width of the text area.

The *Straddle* setting has no effect on tables in a single-column document. In a single-column document, if a table is wider than the column, it soft-straddles. A **soft straddle** is the same as a straddle, but the table does not need a *Straddle = Yes* specification.

**Table Widow and  
Orphan Control  
Properties**

In a single-column document, *Orphan Control* and *Widow Control* settings determine the number of rows in a table that can appear at the top and bottom of a page. In a multicolumn document, these settings determine the number of table rows that can appear at the top and bottom of a column. You can set a value from 1 to 16.

**Table Border  
Properties**

The *Border Visible* properties determine whether table borders are visible.

The *Border Weight* properties determine the thickness of the table borders. The thickness increases from 0.125 to 6.125 points in increments of 0.125 points. Double, which is optional, makes double borders around the table. When *Border Weight* is specified, *Border Visible* is assumed to be Yes.

The *Border Color* properties determine the color of the borders. The values are the colors defined in the `<!Color Definitions....>` declaration for the document (see *ASCII Format for Text*). The range for color designation codes is 0 to 255.

**<Master Row, ...>Declaration**

*<Master Row, ... >* is a declaration, since it creates a master. Each table has its own set of master rows.

Unlike other declarations, which use the *<!... >* format, the master row declaration is enclosed only between angle brackets.

The markup for a typical master row is as follows:

```
<Master Row, name,
    A-Page = Yes/No,
    Allow Page Break After = Yes/No,
    Allow Page Break Before = Yes/No,
    Allow Page Break Within = Yes/No,
    Begin New Column = Yes/No,
    Begin New Page = Yes/No,
    Bottom Margin = x inches [0.0266 inches],
    Top Margin = x inches [0.0266 inches],
    Font = font,
    Header = Yes/No,
    Footer = Yes/No,
    Read Only = Yes/No,
    Border = Yes/No,>
```

**Row A-Page**

The *A-Page = Yes/No* property is for internal use only; do not specify it.

**Row Page Break**

The *Begin New Page* and *Allow Page Break After* properties determine whether a row should begin a new page and whether a page break can occur after the row. *Allow Page Break Before* determines whether a page break can occur before the row. *Allow Break Within* permits spillover of table cell data onto the following page. The *Begin New Column* property in a multicolumn document indicates whether a row should begin a new column. You use these properties with rows the same way you use them with components.

**Row Margin**

The top margin of a row is the area just below the row's top ruling; the bottom margin is the area just above the row's bottom ruling. If you change the weight of a ruling, the size of the row margins does not change.

**Row Font**

If a row has a revision bar, the *Font* property determines the font used in it.

.....

**Row Header and Footer**

The row *Header* property determines whether a row is the running header for a table. The row *Footer* property determines whether a row is the running footer for a table.

**Border**

The *Border* property specifies whether the row has border rulings.

**<Table, ...> Command**

The <*Table*, ...> command properties are the same as the properties on the <!*Master Table*, ...> declaration. The name of the table must be the first field in the ASCII markup for the <*Table*, ...> command. The properties for an instance of a table come from the master table; for properties not specified on the master, the instance takes the system defaults for tables. For a list of defaults refer to Appendix A, *Default Values for ASCII Format Documents*.

**<Row, ...> Command**

The properties for the <*Row*, ...> command are identical to the properties on the <!*Master Row*, ...> declaration. Refer to *The <!*Master Row*, ...> Declaration* in this chapter for details. *Hidden* and *Read-Only* are special properties that can be declared for row instances.

**Hidden = Yes/No**

On the row instance, you can specify user-defined attributes and values and you can hide entire rows, depending on the control expression in effect. If the row is hidden according to the control expression in effect, the row has a specification of *Hidden* = *Yes* on output. The *Hidden* property is for output only; the loader ignores the *Hidden* property on input.

These properties are specified on the row instance as follows:

```
@attribute = value,
Hidden = Yes,
```

Both *Hidden* = *Yes* and *Hidden* = *No* are ignored by the Interleaf ASCII loader.

**Read Only**

*Read Only* = *On* freezes the contents of the row. No editing operations may be performed until this property is set to *Off*.

**<Cell, ...> Command**

The markup for *<Cell, ...>* is as follows:

```
<Cell,
    @attribute = value
    Auto Edit = Yes/No,
    Top Ruling Weight = 0.125 - 6.125 Double,
    Top Ruling Color = 0-255,
    Top Ruling Visible = Yes/No,
    Left Ruling Weight = 0.125 - 6.125 Double,
    Left Ruling Color = 0-255,
    Left Ruling Visible = Yes/No,
    Size Contents To Width = Yes/No,
    Straddle = n [1],
or
    Straddle = vertical,
    Vertical Alignment = Top/Bottom/Center,
    Color = 0-255,
    Pattern = 0-255,
    Not Selectable = Yes/No,
    Read Only = Yes/No>
```

You can use a *<Frame, ...>* command in place of a *<Cell, ...>* command; it is especially useful when you have a graphics object within a cell. In Figure 3-2, the last cell is a frame containing a graphics object.

---

**Note**

You can change a cell's microdocument page properties (the equivalent of making changes to the microdocument's Page Properties dialog box) by including a *<!Page,...>* declaration Within a *<Cell, ...>* command. For example, the following command,

```
<Cell> <!Page,
    Hyphenation= off>
<"2x2:cell">
```

turns off hyphenation on the microdocument Page Properties dialog box for this cell. For more information about the *<!Page...>* declaration, refer to the chapter *ASCII Markup for Text*.



---

**Note**

If you use a *<Frame, ...>* command in place of a cell, you can use the following additional markup as well as the property values used in markup for *<Cell, ...>*. The results will be unpredictable, or a crash may occur, if you use options other than the regular cell markup and the following in the additional markup for *<Frame,...>*:

```
<Frame, ...>:
Size Contents to Height = No,
Vertical Alignment = Top,
```

**Cell Auto Edit**

To include graphics in a cell, invoke the Graphics Editor by setting *Auto Edit* to No. By default the setting for *Auto Edit* is Yes, which invokes the Object Editor.

**Cell Ruling**

The *Left Ruling Visible* and *Top Ruling Visible* properties determine whether the left and top cell rulings are visible. The *Left Ruling Weight* and *Top Ruling Weight* properties control the weight of the left and top rulings of a cell and whether these rulings are double rulings. The range of weights is 0.125 to 6.125 points in increments of 0.125 point.

The *Left Ruling Color* and *Top Ruling Color* properties control the color of the left and top rulings of a cell. At the intersection of rulings, the visibility, color, and weight of the horizontal ruling takes precedence over the settings for the vertical ruling.

**Cell Straddle**

The cell *Straddle* property controls how many columns a cell can straddle horizontally. The default is 1.

In Interleaf 7, cells may also straddle vertically, although the default is horizontal. See *Sample Table* in this chapter for an illustration of the markup for vertical straddles.

**Cell Vertical Alignment**

The *Vertical Alignment* property determines the vertical alignment of the data in a cell. The values are Top, Bottom, and Center.

**Cell Background Color and Pattern**

Background color and pattern information may be included in markup. Numerical values for color and pattern assignment are given in *<Color Definitions, ...>* in the chapter *ASCII Format for Text*.

**Cell Not Selectable** If *Not Selectable* is set to Yes for a cell or set of cells, they cannot be selected interactively with Interleaf 7. A setting of Yes in effect freezes the contents of the cells to editing operations.

**Cell Read Only** *Read Only = On* transforms a cell into a read-only cell.

**Sample Tables**

The ASCII markup for a basic table is straightforward, but table markup can become complex.

The following is an example of markup for a simple table:

```
<Table, tablename ..., Columns = 3, Column 1 Width = 1 inch, ...>
<Master Row, master_row_name>
<Cell> .... <Cell> ... <Cell> ...
<Row, row_name>
<Cell> ... <Cell> ... <Cell> ...
<End Table>
```

Table markup must conclude with the *<End Table>* command.

**Sample 1: Empty 3x3 Table** Figure 3-1 shows an empty 3-by-3 table created in a generic empty document using the Create→ *<Table>*→ *<Numeric>* command.


Figure 3-1. Empty table.

The following are the Master declarations for the table in Figure 3-1:

```
<!Master Table, "3x3",
    Columns =                3,
    Column 1 Width =        1 units,
    Column 2 Width =        1 units,
    Column 3 Width =        1 units,
    Row =                    "row",
    Row =                    "row",
    Row =                    "row">

<Master Row, "row">

<Cell><"3x3:cell",
    Hidden =                  yes>

<Cell><"3x3:cell",
    Hidden =                  yes>

<Cell><"3x3:cell",
    Hidden =                  yes>

<End Table>
```

Table instance markup for the table in Figure 3-1 begins here:

```
<Table, "3x3",
      Columns =           3,
      Column 1 Width =   1 units,
      Column 2 Width =   1 units,
      Column 3 Width =   1 units>

<Master Row, "row">

<Cell><"3x3:cell",
      Hidden =           yes>

<Cell><"3x3:cell",
      Hidden =           yes>

<Cell><"3x3:cell",
      Hidden =           yes>

<Row, "row">
<Cell><"3x3:cell">

<Cell><"3x3:cell">

<Cell><"3x3:cell">

<Row, "row">
<Cell><"3x3:cell">

<Cell><"3x3:cell">

<Cell><"3x3:cell">

<Row, "row">
<Cell><"3x3:cell">

<Cell><"3x3:cell">

<Cell><"3x3:cell">

<End Table>
```

**Sample 2:  
Complex Table**

The sample table in Figure 3-2 has more complex markup, including rulings of various types, vertical and horizontal straddle cells, and graphic objects within a cell. Following the table is the relevant portion of the ASCII markup for a document containing this table.


The two columns directly below have proportional width; that is, their width will adjust proportionally to the entire width of the table if any change in width is made in any other column or margin.		This column has fixed width.
centered and vertically straddled	This row illustrates vertical alignment. This cell is top aligned (the default). The other cells are centered and bottom aligned.	The cell below has color fill.
	The ruling above this cell is double width.	
This footer row straddles 2 columns. The cell to the right has a diagram:		

Figure 3-2. Sample table

The following are the Master declarations for the table in Figure 3-2:

```
<!Master Table, "3x3",
    Columns = 3,
    Top Margin = 0.09 Inches,
    Bottom Margin = 0.09 Inches,
    Top Border Weight = 2,
    Bottom Border Weight = 2,
    Left Border Weight = 2,
    Right Border Weight = 2,
    Column 1 Width = 1 Inches,
    Column 2 Width = 4 Inches,
    Column 3 Width = 1 Inches,
    Row = "title",
    Row = "head",
    Row = "row",
    Row = "row",
    Row = "row",
    Row = "row",
    Row = "row",
    Row = "row",
    Row = "foot">
<Master Row, "foot",
    Top Margin = 0.03 Inches,
    Bottom Margin = 0.03 Inches,
    Footer = yes>
```

```

<Cell, Straddle = 2><"3x3:foot",
    Hidden = yes,
    Font = F16@Z7,
    Line Spacing = 1.0435 lines>
<Cell><"3x3:foot",
    Hidden = yes,
    Font = F16@Z7,
    Line Spacing = 1.0435 lines>
<Master Row, "row",
    Top Margin = 0.03 Inches,
    Bottom Margin = 0.03 Inches>
<Cell><"3x3:left",
    Hidden = yes,
    Font = F16@Z7,
    Line Spacing = 1.0435 lines>
<Cell><"3x3:middle",
    Hidden = yes,
    Font = F16@Z7,
    Line Spacing = 1.0435 lines>
<Cell><"3x3:right",
    Hidden = yes,
    Font = F16@Z7,
    Line Spacing = 1.0435 lines>
<Master Row, "title",
    Top Margin = 0.03 Inches,
    Bottom Margin = 0.03 Inches,
    Allow Page Break After = no,
    Header = yes>
<Cell, Straddle = 3><"3x3:title",
    Hidden = yes,
    Font = F22@Z7,
    Line Spacing = 1.1205 lines>
<End Table>

```

Table instance markup begins here:

```

<Table, "3x3",
    Columns = 3,
    Top Margin = 0.12 Inches,
    Bottom Margin = 0.12 Inches,
    Left Margin = 1 Inches,
    Allow Page Break Within = no,
    Top Border Weight = 4,
    Bottom Border Weight = 4,
    Left Border Weight = 4,
    Right Border Weight = 4,
    Header Border Weight = 4,

```

```

Footer Border Weight = 4,
Column 1 Width =      1 units,
Column 2 Width =      4 units,
Column 3 Width =      0.95 Inches>

<Master Row, "title",
    Top Margin =      0.03 Inches,
    Bottom Margin = 0.03 Inches,
    Allow Page Break After =      no,
    Header =          yes>

<Cell, Straddle = 3,
    Top Ruling Weight =      1.5,
    Left Ruling Weight =      2><"3x3:title",
    Hidden =                  yes>

<Master Row, "row",
    Top Margin =      0.03 Inches,
    Bottom Margin = 0.03 Inches>

<Cell,
    Top Ruling Weight =      1.5,
    Left Ruling Weight =      2><"3x3:left",
    Hidden =                  yes,
    Line Spacing =            1.1044 lines>

<Cell,
    Top Ruling Weight =      1.5,
    Left Ruling Weight =      2><"3x3:middle",
    Hidden =                  yes,
    Line Spacing =            1.1044 lines>
<Cell,
    Top Ruling Weight =      1.5,
    Left Ruling Weight =      2><"3x3:right",
    Hidden =                  yes>

<Master Row, "foot",
    Top Margin =      0.03 Inches,
    Bottom Margin = 0.03 Inches,
    Footer =          yes>

<Cell, Straddle = 2,
    Top Ruling Weight =      1.5,
    Left Ruling Weight =      2><"3x3:foot",
    Hidden =                  yes>

```

```
<Row, "title"><|,"1">
```

```
<Cell, Straddle = 2,
    Top Ruling Weight =      1.5,
    Left Ruling Weight =     2><"3x3:title",
    Font =                  F8@Z7,
    Line Spacing =          1.1004 lines>
```

The two columns directly below have proportional width; that is,<HR> their width will adjust proportionally to the entire width of the table if any change in <SR> width is made in any other column or margin.

```
<Cell,
    Top Ruling Weight =      1.5,
    Left Ruling Weight =     2><"3x3:title",
    Font =                  F8@Z7,
    Line Spacing =          1.1004 lines>
```

```
this column <SR>
has fixed <SR>
width
```

```
<Row, "row">
<Cell, Vertical Alignment = Center,
    Top Ruling Weight =      1.5,
    Left Ruling Weight =     2><"3x3:left",
    Line Spacing =          1.1044 lines>centered and <SR>
vertically <SR>
straddled
```

```
<Cell,
    Top Ruling Weight =      1.5,
    Left Ruling Weight =     2><"3x3:middle",
    Line Spacing =          1.1044 lines>
```

This row illustrates vertical alignment. This cell is top aligned <SR> (the default). The others are centered and bottom aligned.

```
<Cell, Vertical Alignment = Bottom,
    Top Ruling Weight =      1.5,
    Left Ruling Weight =     2><"3x3:right",
    Alignment =              Outer>
```

```
The cell <HR>
below has<HR>
color fill.
```

.....



```
<Row, "row">
<Cell, Straddle = Vertical, Vertical Alignment = Center,
      Top Ruling Weight =      1.5,
      Left Ruling Weight =      2>
```

```
<Cell,
      Top Ruling Weight =      1.5 double,
      Left Ruling Weight =      2><"3x3:middle",
      Line Spacing =          1.1044 lines>
```

The ruling above this cell is double width.

```
<Cell,
      Color = 5,
      Top Ruling Weight =      1.5,
      Left Ruling Weight =      2><"3x3:right"><Row, "foot">
```

```
<Cell, Straddle = 2,
      Top Ruling Weight =      1.5,
      Left Ruling Weight =      2><"3x3:foot">
```

This footer row straddles 2 columns. The cell to the right has a diagram:

```
<Frame,
      Top Ruling Weight =      1.5,
      Left Ruling Weight =      2,
      Diagram =
V8,
(g9,1,0
(T15,1,0,-0.04,0,0,127,0,0,127,1,0,3,
<!Page, Width = 0.9116813 Inches, Height = 0.13837 Inches>
<"3x3:foot">
```

```
<End Text>)
```

```
(e9,2,0,0.1135718,0.127334,0.7181104,0.127334,0.1135718,0.006
0002,7,127,5,7,0,1,0)
```

```
(E16,0,0,5,1,1,0.0533333,1,15,0,0,1,0.04,0,0,1,7,127,7,0,0,7,
0,1,1,0.0666667,0.0666667,6,6,0,0.0666667,6))>
<End Table>
```

.....

# *ASCII Format for Graphics Objects*

# 4

This chapter describes the Interleaf ASCII format for graphics objects, formerly called diagramming objects. Despite this change in terminology, the term *diagramming* is still used in the Interleaf ASCII markup for graphics objects.

This chapter describes general properties, such as locks, edge, and fill, that apply to all graphics objects. It also describes the object-specific markup for each graphics object. These fields represent the properties of the graphics object.

Graphics objects in Interleaf 7 include lines, boxes, arcs, Béziers (Bézier curves), splines, text strings, microdocuments, outline text, charts, raster images, and OLE (object linking and embedding) objects. These objects always appear in a frame.

---

**Note**

Starting with Release 5.2, components and graphics objects can have Interleaf Lisp statements attached upon output. Changes affecting graphics objects are described in *ASCII Lisp Method Storage for Graphics Objects* in this chapter.

## General Markup Rules

The markup for graphics objects always appears in a `<Frame, ...>` command. Except for header and footer frames, the markup for frames containing graphics objects is similar to this:

```
<Frame,
    Name = Inline,
    Placement = At Anchor,
    Width = .41 inches,
    Height = .137 inches,
    Diagram =
V11,
    (g9,0,0,
      (e8,16,0,,0.4,2.1,2.0,1.1,0.3,1.0, 0, 0, 5, 7, 0, 6, 0) )>
```

Following the frame property specifications, information for each diagram begins with the uppercase letter *V*, followed by the diagram version number (*11* in the example) and a comma. The graphics objects follow the version number. You can have any number of graphics objects in a frame.

All objects in a frame are considered part of a top-level group. In the example, there is a group object beginning with the letter *g* and an ellipse object beginning with the letter *e*. The top-level group can contain other groups.

The markup for graphics objects obeys the following syntax:

- a left (opening) parenthesis
- a letter indicating object type—for example, *g* for group
- an object version number—for example, *9* for version 9
- a comma
- the *z* coordinate (an integer)
- a comma

- the general graphic object flags field (32-bit, unsigned, written in decimal)

starting with Release 5.2 and the V11 diagram version number, the first field following the flags field is for Interleaf Lisp data. If no Lisp is present, the field is empty. For more information, refer to *ASCII Lisp Method Storage for Graphics Objects* in this chapter.

- a series of other comma-enclosed fields

The number of fields and the type of fields depend on the object type.

- a right (closing) parenthesis

## Object Type and Version

All graphics objects have two identifiers: a type letter (such as *c*, *g*, *N*, or *O*) that indicates type of graphics object, and a version number that identifies the current version of the graphics object.

Valid object types with their type letters and version numbers are

- group (g9)
- polygon (p8)
- arc (a9)
- spline (S14)
- text string (t14)
- microdocument (T15)
- line (v7)
- plotter (V9)
- chart (c5)
- image (i18)
- Encapsulated PostScript object (n6)
- ellipse (e9)
- edit state object (E16)
- equation object (m9)

.....

- convert-to-outline object (o4)
- named graphics object (N10)
- Bézier object (z5)
- outline object (C6)
- OLE object (q6)

---

**Note**

Letters designating object type are case-sensitive.

The equation object (m9) and the outline object (C6) are not documented.

## Layering

Front/back positioning is determined by the z coordinate. Overlapping objects with higher z values occlude those with lower ones.

## Locks

The general graphics object flags field contains lock bits. It is a 32-bit unsigned integer, written in decimal. A zero, indicating that the lock is not set, suffices in most cases.

The following list shows the mapping of the bits in the general flags field; bit 0 is the least significant bit.

Bit	Function
0	no flag bits set
1	lock size
2	lock position
3	lock against rotation
4	lock grouping on
5	lock against selection
6	lock line widths
7	lock fill pattern
8	lock font
9	lock against printing
10	lock against cutting
11	lock gravity off
12	lock control points off
13	lock aspect ratio
14	lock smoothness
15	reserved for system use
16	reserved for system use

The following bits are discarded if the object is being saved with version 3:

17	lock object stickiness
18	reserved for system use
19	reserved for system use
20	reserved for system use
21	lock dash pattern
22	lock fill color
23	lock fill transparency
24	lock edge color
25	lock edge transparency
26	object is hidden
27	pass messages on to Lisp
28	object queued for incremental redisplay
29	GenFree called; free during incremental redisplay
33	number of flag bits + 1

## Edge and Fill Properties

The edge properties consist of four fields: visibility, color, weight, and dashes. The fill properties are made up of three fields: visibility, color, and pattern.

The visibility field for edge and fill properties can be either 0 (visible) or 127 (invisible).

The edge weight field is expressed in printers' points. The minimum legal width is 0.125; the maximum is 6.125. If a specified edge weight is too large or too small, it is converted to the nearest legal value on input.

Figure 4-1 shows the line pattern values and the lines they produce.

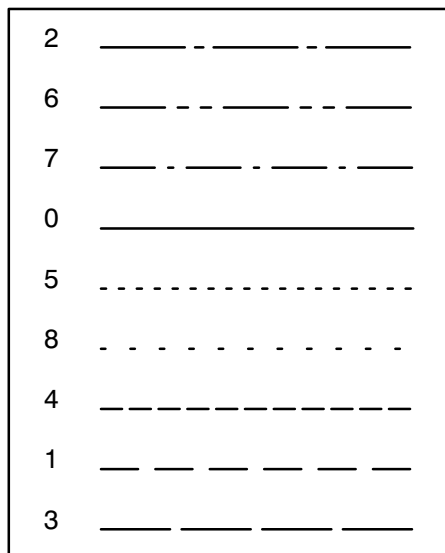


Figure 4-1. Line pattern values and the lines they represent.

For more information on color and pattern, refer to the sections on the `<!Colors, ...>` declaration and the `<!Patterns, ...>` declaration in the chapter *ASCII Format for Text*.

## Graphics Error Messages

If the loader finds an error in a diagram when loading a document, it displays a warning message such as: *ASCII Loader Warning on line 29: Error in loading diagram* .

The loader skips the contents of the frame and loads an empty frame. If the loader encounters more than one error in the diagram, it displays only the first message. For more information about error messages, refer to Appendix B, *ASCII Format Error Messages*.

## ASCII Lisp Method Storage for Graphic Objects

All graphics objects have a field after the locks or flags field for storing Lisp code with the object. If no Lisp code is present, there will be an empty field as shown in the following example:

```
V11,
(g9,1,0,
 (v7,1,0,,1.0666667, [...])
```

For both the group (g9) and the vector (v7), 1 represents the z coordinate and 0 is the locks field. Following each is an empty Lisp field. In the case of the group, the Lisp field occurs between the comma and the vector's left (opening) parenthesis. The Lisp field for the vector occurs between the two commas.

If the diagramming lock bit (value 0x04000000, decimal 67108864) is not set, the Lisp field for the group must be empty. The field is named `:allow-lisp-override` from Lisp.

With the bit set, executing the following Lisp code:

```
(tell obj mid:set-props :locks '(:allow-lisp-override))
(tell obj mid:put-saved-data :foo "(<>")
```

results in the following markup:

```
V11,
(g9,1,0,
 (v7,1,67108864,\(tell\ *load-object*\ mid:put-data
 \ 'ileaf::saved-data\ '(:foo\"(\)\(<>\>\"\\)\x0a\),
 1.0666667,0.7333333,1.9333333,1.3333333,7,0,1,0)
 (E16,0,0,,5,1,1,0.0533333,1,15,0,0,1,0,0,0,1,5,127,
 7,0,0,7,0,1,1,0.0666667,0.06
 66667,6,6,0,0.0666667,6))>
```

For information on how Lisp output affects the text object system, refer to the chapter *ASCII Format for Text*.



---

**Note**

The ASCII loader currently does not properly handle Lisp symbols split across line endings in diagramming markup. To avoid this problem, keep saved Lisp data expressions within diagramming objects as short as possible.

---

## Markup for Specific Objects

The coordinates of objects in the descriptions in this chapter are in inches relative to the upper-left corner of the containing frame. Positive x coordinate values go to the right from the upper-left corner of the containing frame; positive y coordinate values go down from the upper-left corner of the containing frame. All frames are rectangular.

When you declare a value, do not use more than nine digits to the left of the decimal point and six digits to the right of the decimal point. Because the internal floating-point precision is roughly nine digits, Interleaf recommends that you use no more than a total of nine significant digits, left and right, to specify each coordinate.

## Lines

In markup, a lowercase *v* (for *vector*) specifies a line segment, and the version number is 7. Here is an example of markup for a line segment:

```
(v7,1,0,,0.7,1.3,2.1,1.3,7,0,6,0)
```

Value	Represents
v7	object type: line (version 7)
1	z coordinate
0	general graphic object flags
<i>empty field</i>	saved lisp data (none in this example)
0.7, 1.3	x,y coordinates of the beginning (in inches)
2.1, 1.3	x,y coordinates of the end (in inches)
7	color (black)
0	transparency (opaque; visible)
6	line width (points)
0	line pattern (solid)

## Groups

A group is a collection of objects treated as a single unit.

In markup, a lowercase *g* represents a group, and the version number is 9. The *z* coordinate and flags follow *g9*. The rest of the group consists of a list of objects.

For example, the markup for a group of four line segments might look like this:

```
(g9,1,0,
  (v7,1,0,,0.7,1.3,2.1,1.3,7, 0, 6, 0)
  (v7,2,0,,0.9,1.7,3.1,2.3,7, 0, 6, 0)
  (v7,3,0,,0.3,4.3,2.6,1.9,7, 0, 6, 0)
  (v7,4,0,,1.7,1.4,4.1,2.3,7, 0, 6, 0))
```

All graphics objects in a frame are considered part of a group. For example, the markup for a frame containing a single line segment might look like this:

```
<Frame ...
  v11,
  (g9,1,0,
    (v7,1,0,,1.7,1.4,4.1,2.3,7, 0, 6, 0))>
```

The markup for a frame containing a single line segment and a group of three others might look like this:

```
<Frame ...
  v11,
  (g9,1,0,
    (v7,1,0,,1.7,1.4,4.1,2.3,7, 0, 6, 0
      (g9,2,0,
        (v7,2,0,,0.7,1.3,2.1,1.3,7, 0, 6, 0)
        (v7,3,0,,0.9,1.7,3.1,2.3,7, 0, 6, 0)
        (v7,4,0,,0.3,4.3,2.6,1.9,7, 0, 6, 0))))>
```

The first *g* is not significant for designating a group. This top-level group object contains all the objects in the frame. The top-level group can contain other nested groups.

Here is the markup for a more complex group, a polygon with an ellipse (visually) inside:

```
(g9,2,0,
  (p8,2,8,,5,5,127
    (g9,2,0,
      (v7,2,65536,,2.133333,2.8,7.8,2.8,7,0,6,0)
      (v7,3,65536,,7.8,2.8,7.8,1.066667,7,0,6,0)
      (v7,4,65536,,7.8,1.066667,2.133333,1.066667,7,0,6,0)
      (v7,5,65536,,2.133333,1.066667,2.133333,2.8,7,0,6,0))))
(e9,6,0,,3.666667,1.2,7.266667,1.2,3.666667,2.533333,5,127,5,7,0,6,0))
```

Note

As with other diagramming objects, the group (g9) markup includes a field for Interleaf Lisp data. In the case of the group, the Lisp field occurs between the comma at the following third field and the left (opening) parenthesis for the first object in the field. Because this field is often empty, it may be easy to overlook. For example, in the markup above, if there were saved Lisp data on the group it would appear following g9,2,0, on the first line.

Polys

In markup, a lowercase *p* specifies a poly, and the version number is 8. Here is an example of markup for a simple poly, a box:

```
(p8,2,8,,5,7,127
  (g9,2,0,
    (v7,2,65536,,2.866667,7.066667,7.066667,7.066667,7,0,6,0)
    (v7,3,65536,,7.066667,7.066667,7.066667,4.666667,7,0,6,0)
    (v7,4,65536,,7.066667,4.666667,2.866667,4.666667,7,0,6,0)
    (v7,5,65536,,2.866667,4.666667,2.866667,7.066667,7,0,6,0))))
```

Value	Represents
p8	object type: poly (version 8)
2	z coordinate
8	flags
<i>empty field</i>	saved lisp data (none in this example)
5	fill pattern (solid)
7	fill color (black)
127	fill transparency (invisible)
g9 ...	a group of four line segments making a 3- by 1-inch box

**Note**

The end of the first element in a poly (second x,y pair) and the beginning of the second element (first x,y pair) must match, and the end of the second element must match the beginning of the third element, and so on.

**Ellipses**

An ellipse is defined by specifying a bounding parallelogram. In Figure 4-2, the parallelogram, and consequently the ellipse, is uniquely defined by giving the coordinates of A followed by those of B and C.

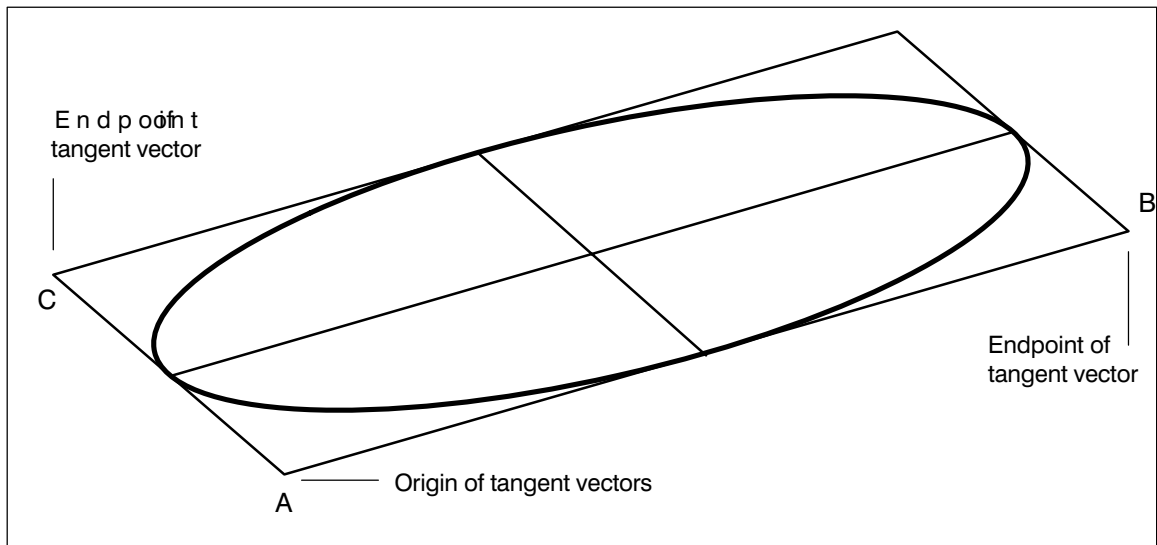


Figure 4-2. An ellipse.

In markup, a lowercase *e* represents an ellipse object, and the version number is 8. Here is an example of markup for an ellipse:

```
(e8,16,0,,0.4,2.1,2.0,1.1,0.3,1.0,0,0,5,7,0,6.125,0)
```

Value	Represents
e9	object type: ellipse (version 9)
16	z coordinate
0	general graphic object flags
<i>empty field</i>	saved lisp data (none in this example)
0.4, 2.1	x,y coordinates of origin of tangent vectors (in inches)
2.0, 1.1	x,y coordinates of tangent vector endpoint (in inches)
0.3, 1.0	x,y coordinates of tangent vector endpoint (in inches)
0	interior color (white)
0	interior transparency (opaque)
5	interior pattern (solid)
7	border color (black)
0	border transparency (opaque)
6.125	border width (points)
0	line pattern (solid)

## Arcs

Interleaf 7 supports the three types of conic arcs: elliptic, parabolic, and hyperbolic.

Arcs have a six-point model:

- *Point 1* is the tangent at the beginning point.
- *Point 2* is the beginning point.
- *Point 3* is a point on the arc between *point 2* and *point 4*.
- *Point 4* is a point on the arc (usually the endpoint).
- *Point 5* is the tangent at *point 4*.
- *Point 6* is the endpoint.

In a hyperbolic arc, all the points must be on the same half of the hyperbola.

If an elliptic arc is greater than 180 degrees, the angular separation of the tangent vectors cannot be less than  $\arctan(2)$ , or approximately 63.5 degrees. For closed, or almost closed, arcs, *point 4* will be distinct from *point 6* to satisfy this condition.

In Figure 4-3, *points 4 and 6* are the same on the non-extended arc, and distinct on the extended arc.

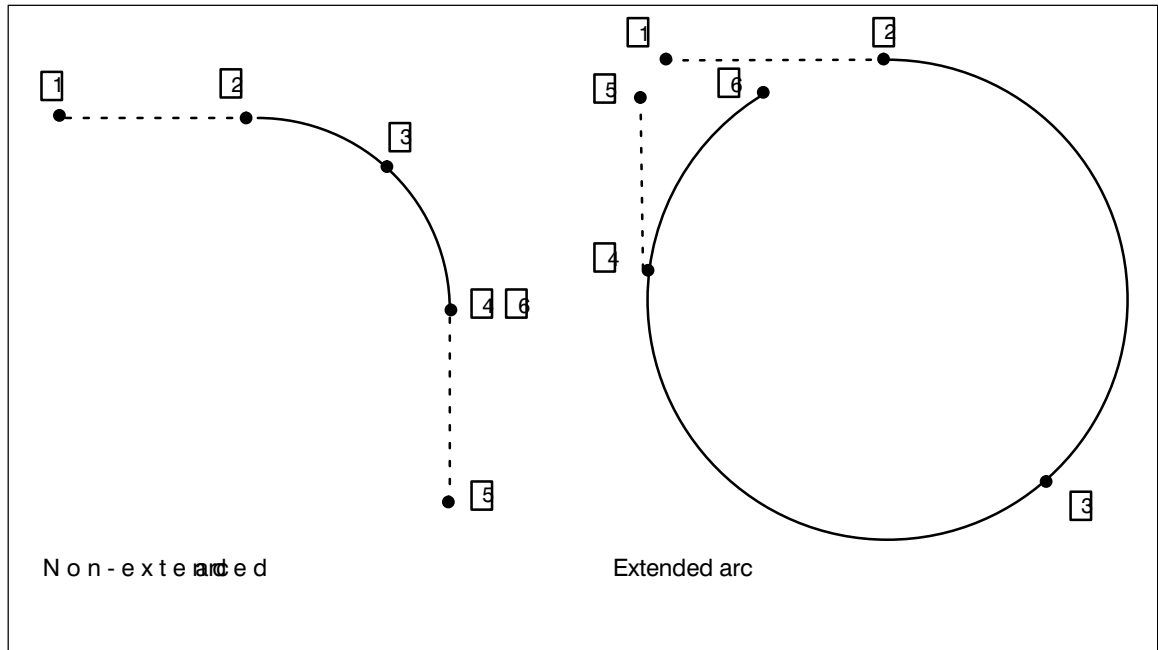


Figure 4-3. Elliptic arcs.

In markup, a lowercase *a* represents an arc, and the version number is 9. The z coordinate and flags follow *a9*. The number 1 is the format; 1 is the only valid value.

Here is an example of markup for a non-extended elliptic arc:

```
(a9,1,0,,1,0.5,1,1.5,1,2.207107,1.292894,2.5,2,2.5,3.0,2.5,2,5,7,127,7,0,6,0)
```

Value	Represents
a9	object type: arc (version 9)
1	z coordinate
0	general graphic object flags
<i>empty field</i>	saved lisp data (none in this example)
1	format (Interleaf)
0.5,1	x,y coordinates of point 1 (tangent at beginning point)
1.5,1	x,y coordinates of point 2 (beginning point)
2.207107,	
1.292894	x,y coordinates of point 3 (point between point 2 and point 4)
2.5,2	x,y coordinates of point 4 (endpoint)
2.5,3.0	x,y coordinates of point 5 (tangent at point 4)
2.5,2	x,y coordinates of point 6 (same as point 4)
5	interior pattern
7	interior color
127	interior transparency
7	border color
0	border transparency
6	border width
0	border pattern

Here is an example of markup for an extended elliptic arc:

```
(a9,1,0,,1,3.0,1,4,1,4.25,2.96,3.03,1.74,3.29,0.77,3.5,1.13,5,7,127,7,0,6,0)
```

Value	Represents
a9	object type: arc (version 9)
1	z coordinate
0	general graphic object flags
<i>empty field</i>	saved lisp data (none in this example)
1	format (Interleaf)
3.0,1	x,y coordinates of point 1 (tangent at beginning point)
4,1	x,y coordinates of point 2 (beginning point)
4.25,2.96	x,y coordinates of point 3 (a point on the arc between point 2 and point 4)
3.03,1.74	x,y coordinates of point 4 (a point on the arc)
3.29,0.77	x,y coordinates of point 5 (tangent at point 4)
3.5,1.13	x,y coordinates of point 6 (endpoint)
5	interior pattern
7	interior color
127	interior transparency
7	border color
0	border transparency
6	border width
0	border pattern

## Splines

Splines are smooth curved lines drawn through or near a series of control points called **knots**.

In markup, an uppercase *S* specifies a spline object, and the version number is *14*.

The z coordinate and flags fields follow the *S14*. The coordinates of the basis of the spline are next, followed by color, pattern, transparency, and the total number of knots.

Each ensuing knot consists of its x,y coordinates in inches, the line color and transparency, the line width, and the line pattern. After the last knot, there is one more field consisting of the letter *O* or *C* depending on whether the spline is open or closed. If the spline is open, you must specify the endpoints in duplicate.



Here is an example of markup for an open spline:

```
(S14,8,0,,3.2,6.86,5,5,127,11,0,0,7,0,16,0,0,0,7,0,16,0,2.8,2.533333,
  7,0,16,
0,-1.333333,1.266667,7,0,16,0,-1.874191,1.10092,7,0,16,0,0.066667,
  0.666667,7,0
,16,0,0,0.6,7,0,16,0,-0.533333,0.066667,7,0,16,0,-0.866667,-0.4,7,0,
  16,0,-1.19
5464,-0.860317,7,0,16,0,-1.195464,-0.860317,7,0,16,0,0)
```

Value	Represents
S14	object type: spline (version 14)
8	z coordinate
0	general graphic object flags
<i>empty field</i>	saved lisp data (none in this example)
3.2,6.86	x,y coordinates (inches) of basis of spline
5	interior pattern
5	interior color
127	interior transparency
11	number of knots
0, 0	x,y coordinates (inches) of first knot relative to basis
7	segment color
0	segment transparency
16	segment width in 1/8 screen pixels (1/6000ths of an inch)
0	segment pattern
0, 0	x,y coordinates (inches) of second knot relative to basis
7	segment color (second knot)
0	segment transparency (second knot)
16	segment width (second knot)
0	segment pattern (second knot)
... etc.	
O	spline is open (end does not meet beginning)

Text Strings

A text string is a simple object without most of the characteristics of a formatted microdocument.

In markup, a lowercase *t* represents a text string object, and the version number is *14*.

Here is an example of markup for a text string:

(t14,15,0,,6.6,5.6,1,7,0,180,us,wst:dutch12text string ...)

Value	Represents
t14	object type: text string (version 14)
15	z coordinate
0	general graphic object flags field
empty field	saved Lisp data (none in this example)
6.6, 5.6	x,y coordinates (inches) of the text anchor (inches)
1	anchor type field
7	line texture or color indicator
0	transparency indicator; visible or invisible
180	angle of rotation of the text string
us	text properties of the text string (underline, kerning, and strikethrough)
wst:dutch12,	character string field specifying the font type
text string ...	the text string

In markup for a text string, each comma-delimited field must occur in order. You do not need to specify the text properties in a particular order.

You can specify any angle of rotation, but Interleaf 7 displays and prints text only if it is non-rotated or is rotated 90, 180, or 270 degrees.

In the text properties field, you can specify text properties for the text string. The markup is identical to regular text except that all properties must be declared in lowercase letters. For example, *u* is used instead of *U* for underlining and *s* is used for *S* (strikethrough). Two commas in sequence indicate default text properties, as shown in the first line of the example markup.

The following examples show specifications for text strings:

```
(t14,1,0,,1.9,2.6,0,7,0,0,,wst:dutch12,a\normal\ text\ string)
(t14,2,0,,1.9,2.9,0,7,0,0,s,wst:dutch12,a\ strike-through\ text\
  string)
(t14,3,0,,1.9,3.0,0,7,0,0,u,wst:dutch12,a\ underlined\ text\
  string)
(t14,4,0,,1.9,3.4,0,7,0,0,p,wst:dutch12,pair\ kerns\ off)
(t14,5,0,,1.9,3.6,0,7,0,0,t1,wst:dutch12,track\ kerns\ tight)
(t14,10,0,,1.9,4.4,0,7,0,0,t6,wst:dutch12,loosest\ track\ kerns)
(t14,11,0,,3.0,6.3,1,7,0,0,,wst:dutch12,This\ is\ unrotated\
  text)
(t14,12,0,,2.96,6.3,1,7,0,90.0,,wst:dutch12,text\ is\ rotated\
  90\ degrees)
(t14,13,0,,3.2,6.5,1,7,0,180.0,,wst:dutch12,text\ is\ rotated\
  180\ degrees)
(t14,14,0,,3.3,6.6,1,7,0,270.0,,wst:dutch12,text\ is\ rotated\
  270\ degrees)
```

## Quoting in Text Strings

In text strings, you use a backslash (\) in place of quotation marks around certain characters. The following characters have special meanings in ASCII format, and you must quote them with a backslash:

< > ( ) , \

To use one of these characters in a text string, precede it with a backslash. Also, use a backslash to quote spaces in a text string. For example, the less-than symbol followed by a space and a comma is:

\< \ ,

Chart labels must follow these graphics object quoting conventions.

## Convert-to-Outline Object

For input only, there is ASCII markup that converts a text string into outline font characters. Interleaf 7 loads this markup and executes a **Convert to Outline** command on the specified text string.

The convert-to-outline markup, like other graphics object markup, must appear within a frame after the version number and specification for the upper-level group.

In markup, a lowercase *o* specifies a convert-to-outline text string, and the version number is 4.

.....

Here is an example of the convert-to-outline markup:

```
(o4,34,12,,0.7,1.2,6.5,wst:swiss16,This\is\ rotated\ text)
```

Value	Represents
o4,	object type ID and version number
34	z coordinate
12	general graphic object flags
<i>empty field</i>	saved Lisp data (none in this example)
0.7, 1.2	x,y coordinate of left baseline (inches)
6.5	rotation angle (radians)
wst:swiss16	character set: font and point size
This\ is\ ...)	text string; the space characters are quoted with backslashes

On input, you specify placement (x,y coordinate of the left baseline of the text string), scaling (point size), and rotation. The specified point size must be an integer, but it does not have to be a point size actually on the system. Interleaf 7 uses the specified point size only for scaling the outline text.

Interleaf 7 does not output the convert-to-outline markup; it outputs only the encrypted outline characters. Since you can specify this markup on input, it is useful for filter programs.

The convert-to-outline markup in the example produces a display of rotated outline font characters as shown in Figure 4-4.

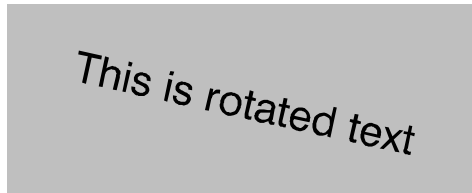


Figure 4-4. A text string converted into rotated outline font characters.

## Microdocuments

In markup, an uppercase *T* indicates that the text object is a microdocument, and the version number is *15*. The z coordinate and flags fields follow the *T15*. There are no special quoting requirements for microdocuments.

A soft-state microdocument has an implicit width of 24 inches. This must be accounted for in the x coordinate of right- or center-aligned microdocuments.

Note

Class definitions for microdocuments can differ if they appear before or after an Include statement. Definitions in an Included file override definitions in the current file. The definition of a class in the Include file takes precedence over a the definition for that class in the current file.

The rest of the markup represents information about the properties of the microdocument. Following the markup for the microdocument object, there must be a newline before the markup for the text itself.

Here is an example of markup for a microdocument containing the sentence “This is a sample microdocument”:

```
(T15,1,12,,3.2,5.9,5,127,5,7,127,6,0,3,
<!Page, Width = 8 inches, Height = 0.133 inches>
<micro:caption,
    Alignment = Left>
This is a sample microdocument
<End Text>)
```

Value	Represents
T15	object type: text-microdocument (version 15)
1	z coordinate
12	general graphic object flags
<i>empty field</i>	saved Lisp data (none in this example)
3.2, 5.9	x,y oordinates of top-left corner of microdocument (in inches)
5	interior color
127	interior transparency
5	interior pattern
7	border color
127	border transparency
6	border width
0	border pattern
3	object-specific flags (for Interleaf internal use except for flag bit 4, which specifies a soft-state microdocument)

The markup for every microdocument has a `<!/Page, ...>` declaration. For a description of the properties and values that you can set on a microdocument, refer to the section on the `<!/Page, ...>` declaration in the chapter *ASCII Format for Text*. The markup for microdocuments must conclude with an `<End Text>` command.

## Plotter or Vector-List Object

In markup, an uppercase *V* specifies the plotter or vector-list object, and the version number is 9. The bounding box parallelogram around the vector-list object is included in the ASCII description. Figure 4-5 shows the coordinate pairs for the bounding box.

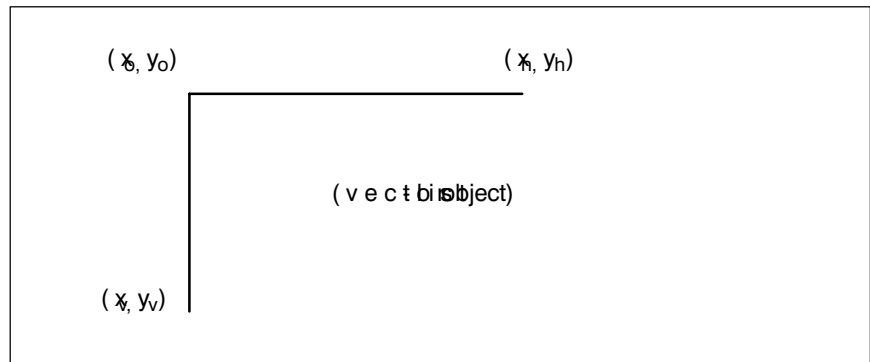


Figure 4-5. Vector-list object with corners of bounding parallelogram labeled.

These coordinate pairs, (xo,yo), (xh,yh), and (xv,yv), are dumped in that order, after color and transparency and before the list of vector sets.

Here is an example of markup for a plotter object:

```
(V9,1,1024,,0,7,0,0,6.480025,0,0,2.967676  
(X0,2,0006)  
(X0,48,8000000000000000000000005EF220002C64A0079801F0000000000713C0  
100360DA300000000000037A4D80002C64A0000CC89)  
(X0,2,0000))
```

Value	Represents
V9	object type: vector-list plotter object (version 9)
1	z coordinate
1024	general graphic object flags
<i>empty field</i>	saved Lisp data (none in this example)
7	color
0	transparency
0,0, 6.480025,0 0,2.967676	specification for the parallelogram that bounds the plotter object; three points, with the fourth point implicit
(X0,2,0006)	beginning of a list of vector sets, each set consisting of two hexafied binary data sets
(X0,2,0000))	end of a list of vector sets, each set consisting of two hexafied binary data sets

Each vector set consists of a variable number of points. There may be a variable number of vector sets encoded in the plotter object.

An ASCII string beginning with "(X0" and ending with ")" encodes binary information in hexadecimal numbers. Each vector set contains two encoded strings: one containing the number of points, and the other containing the points themselves. After the last vector set there is one more hex encoding, (X0,2,0000), indicating that the next vector set has zero points. This indicates that all the vector sets have been encoded.

The first hex set is the number of x,y coordinate pairs in the next object. The maximum number of x,y coordinate pairs is 128 (hex 0080), equivalent to 1024 binary bytes. Hexafication yields a string of 2048 characters. The second set is a series of x,y coordinates in rsu's (ridiculously small units; 1,228,800 rsu's equal 1 inch). After the last vector set, a hex set with two null bytes indicates the end of the list.

A binary encoding starts with an open parenthesis and ends with a close parenthesis. Inside are three fields, delimited by commas. The first always contains "X0"; the second contains, in decimal, the number of bytes that have been encoded. The third contains the encoded information in hexadecimal. The high four bits of each byte are encoded first, followed by the low four bits. This field will contain twice as many characters as are specified in the second field.

The count of points that begins each encoded vector set is, in binary, a 2-byte integer. So (X0,2,0006) indicates that there are six points. (X0,2,001A) indicates 26 points.

Each point consists of two full-word integers, or eight bytes. All the points are encoded in the same string, so following with the six-point example above, the next encoding starts with (X0,48,...).

## Encapsulated PostScript Object

In markup, a lowercase *n* specifies the Encapsulated PostScript (EPS) object, and the version number is 6. The EPS object consists of the text from an EPS file and some data that indicates to Interleaf 7 how to treat the EPS object within the graphics system.

Here is an example of markup for an EPS object (most of the EPS text is not shown):

```
(n6,1,0,,1.1,0,5.29,0,1.1,4.81,130,-491,432,-144
  (p8,1,8552,,5,4,0
    (g9,0,0,
      (g9,0,0,
        (v7,2,0,,1.1,0,5.294444,0,7,0,8,0)
        (v7,3,0,,5.294444,0,5.294444,4.819444,7,0,8,0)
        (v7,4,0,,5.294444,4.819444,1.1,4.819444,7,0,8,0)
        (v7,5,0,,1.1,4.819444,1.1,0,7,0,8,0)))
(X0,2,0800)%!PS-Adobe-1.0\012%%Creator:Adobe\ Illustrator\250 ...)
(X0,2,0800)definefont\ pop}q\012n\012gsave\012%FontEncoding ...)
(X0,2,0295)\ 212\ -279\ c\012227\ -280\ 225\ -296\ 244\ -297\ c\
... )
(X0,2,0000))
```



Value	Represents
n6	object id (n) and version number (6)
1	z coordinate
0	flags
<i>empty field</i>	saved Lisp data (none in this example)
1.1,0	coordinates for upper-left corner (inches)
5.29,0	coordinates for upper-right corner (inches)
1.1,4.81	coordinates for lower-left corner (inches)
130,-491	coordinates for lower-left corner of the original bounding box in the EPS file (points)
432,-144	coordinates for upper-right corner of the original bounding box in the EPS file (points)
(p8,...)	display object (shaded box)
(X0,...)	first in a series of records containing EPS text
(X0,...)	second record
(X0,...)	third record
(X0,2,0000)	terminating record

Each text-containing record consists of a count given by a binary data set converted to hex (see *Plotter or Vector-List Object* in this chapter) followed by a count of the bytes of EPS data. The series of records must be terminated with a zero-length record, (X0,2,0000).

OLE Objects

In markup, a lowercase *q* specifies an OLE object—an object linked or embedded from a Windows-based Object Linking and Embedding (OLE) server application. The version number is 6.

Note

Creation of OLE objects is not supported via ASCII markup, but is described here for debugging purposes only.

Here is an example of markup for an OLE object.

```
(q6,1,0,,0,0.2666667,0.2,2.4000001,2.5333333
(X0,2,0400)
(X0,2,0100)
(X0,4,0400000000)
(X0,4,0000000000)
(X0,4,1000000000)
(X0,16,110200008D0100009B120000A3130000)
(X0,4,1400000000)
(X0,20,0000999900000000001000000FFFFFFFF00000000)
(X0,4,005C0100)
(X0,89088,D0CF11E0A1B11AE10...))
```

Value	Represents
q6	object type: OLE object (version 6)
1	z coordinate
0	general graphic object flags
<i>empty field</i>	saved Lisp data (none in this example)
0	OLE transparency flag (if zero the border of the object is visible)
0.2666667,0.2	x,y coordinates of upper left corner of object
2.4000001,	
2.5333333	x,y coordinates of lower right corner of object
X0,2,0400	first in a series of ASCII renditions of binary values
...	
X0,89088...	last in series

Each ASCII rendition of a binary value starts with (X0, followed by the length of the binary object that was written), then followed by the item itself, where each binary byte is expressed as two hexadecimal ASCII digits, followed by a close parenthesis.

In (X0,2,0400), the length of the binary object is 2. Hex 0x0400 is a short integer that contains the number of items that are to be encoded, which cannot be interpreted until the next line, which indicates whether the integers are written in Intel or in Motorola format. This example uses Intel format; there are four encoded items.

(X0,2,0100) is another binary value. It is 0 for Motorola format and non-zero for Intel format.

(X0,4,040000000) and (X0,4,000000000) are the encoded items. Each represents a pair of encoded binary values. The first value contains, in a fullword, the length of the second value. This seems redundant in ASCII, but is necessary so that versions of Interleaf that do not support OLE (like 6.1.1 on UNIX) can read and write binary OLE format. So here, the first encoded item is 4 bytes long, and it consists of a longword 0.

(X0,4,10000000) and (X0,16,110200008D0100009B120000A3130000) indicate 16 bytes of binary information, and the specification of those bytes.

(X0,4,14000000) and (X0,20,00009999000000000010000000FFFFFFFF00000000) are 20 bytes of binary information.

(X0,4,005C0100) and (X0,89088,D0CF11E0A1B11AE10...) are the next 89,088 bytes of binary information. The last closing parenthesis ends the OLE object.

The four encoded items are considered opaque; even Interleaf doesn't really know much about what is in there. It's OLE data needed to reconstruct information about the OLE object. It is this opacity that prevents OLE objects from being created through the markup.

## Bézier Objects

Bézier objects consist of a series of connected cubic Bézier curves. Each curve has its own edge properties that are associated with the curve's beginning vertex. Figure 4-6 shows a Bézier object with three vertices and associated handles.

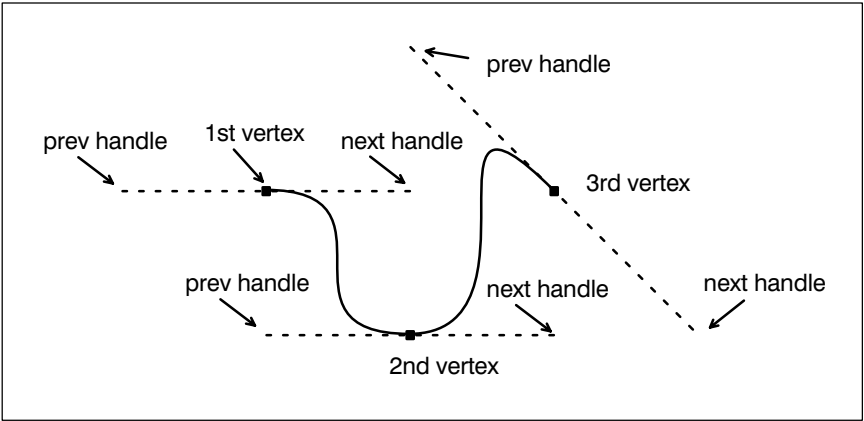


Figure 4-6. A Bézier object with three vertices.

The markup for the Bézier curve in Figure 4-6 is as follows:

```
(z5,1,0,,5,127,5,3,2,1,3,1,1,1,1,7,0,0,1,3,2,4,2,2,2,0,7,0,0,1,4,1,5,2,3,0,2,7,,0,1)
```

Value	Represents
z5	object ID (n) and version number (5)
1	z
0	locks
empty field	saved Lisp data (none in this example)
5	fill color
127	fill transparency
5	fill pattern
3	vertex count
2,1	1st vertex x/y
3,1	next handle x/y
1,1	prev handle x/y
1	Bézier flags (see below)
7	1st curve edge color
0	1st curve edge transparency
0	1st curve edge dashes

1	1st curve edge width
3,2,4,2,2,	
2,0,7,0,0,	2nd vertex...
1,4,1,5,2,3,	
0,2,7,,0,1)	3rd vertex...

*Bézier* flags:

0x0001	first vertex
0x0002	last vertex
0x0004	no curve to next vertex

---

## Named Graphics Objects

The markup for named graphics objects comes in two parts: master definitions, which must appear with other masters (such as components) at the beginning of the document, and instance definitions, which appear within frame markup.

The master definition for named graphics objects is shown in the example given here. The example describes a poly named *my box*. Only markup relevant to named graphics objects is displayed.

```
<!Master Diagramming Object,
    Diagram =
V11,
(N10,0,8,,1796,0,my\box,7,0,1,0,7,127,5,0,0,1.3333333,0,0,0.6,
0,0,1.3333333,0,0
0.6
(p8,0,8,,5,7,127
(g9,0,0,
(g9,0,0,
(v7,0,65536,,0,0,1.3333333,0,7,0,1,0)
(v7,1,65536,,1.3333333,0,1.3333333,0.6,7,0,1,0)
(v7,2,65536,,1.3333333,0.6,0,0.6,7,0,1,0)
(v7,3,65536,,0,0.6,0,0,7,0,1,0)))>
```

With the exception of their special version and number markup, instance definitions for named graphics objects are identical to those for standard graphics objects.

Here is the instance markup for *my box*:

```
<Frame,
  [other markup omitted here]
  Diagram =
V11,
(g9,2,0,
  (N10,2,8,,1796,0,my\
box,7,0,1,0,7,127,5,1.4666667,1.2,2.8,1.2,1.4666667,1.8,0,
,1.3333333,0,0,0.6
  (p8,2,8,,5,7,127
    (g9,2,0,
      (g9,2,0,
        (v7,2,65536,,1.4666667,1.2,2.8,1.2,7,0,1,0)
        (v7,3,65536,,2.8,1.2,2.8,1.8,7,0,1,0)
        (v7,4,65536,,2.8,1.8,1.4666667,1.8,7,0,1,0)
        (v7,5,65536,,1.4666667,1.8,1.4666667,1.2,7,0,1,0))))))>
```

Value	Represents
N10	object type: named graphic object (version 10)
0	z coordinate
8	general graphics object flags
<i>empty field</i>	saved Lisp data (none in this example)
1796	named graphics object flags
my\box	object name. (spaces are permitted—for a name my box write my\ box)
7,0,5,0	edge: color, transparency, width, and pattern
7,0,5	fill: color, transparency, and pattern
1.1333333,1, 2.3333333,1, 1.1333333, 1.866666	xo,yo;xh,yh;xv,yv dimensions of “reference” triangle
0,0,1.2,0,0, 0.8666667	xo,yo;xh,yh;xv,yv dimensions of “transformed” triangle

The flags for named graphics objects are

0x0001	is shared
0x0002	has effectivity attributes
0x0004	edit content directly
0x0010	has valid edge color
0x0020	has valid edge visibility
0x0040	has valid edge dashes
0x0080	has valid edge weight
0x0100	has valid fill color
0x0200	has valid fill visibility
0x0400	has valid fill pattern
0x0800	reserved for system use
0x2000	force hidden

This object has 1796 in its named graphics object flags field, which indicates that it has valid fill pattern, valid fill visibility, valid color, and its contents can be edited directly. (1796 is 704 in hex.)

“Reference” and “transformed” triangles only matter for shared named graphics objects. They are used to record the transformations (such as move, size, and rotate) applied to each instance. For example, when an object is moved, the “transformed” triangle is moved also but the “reference” triangle remains as it was when the instance was created.

When a shared named graphics object instance is loaded or its master’s content is changed, the named graphics object instance is re-created by applying the accumulated transformation to a copy of the master. The edge and fill properties are then applied if valid.

## Edit State Objects

Edit state objects save user preferences per frame if any them differ from the default settings.

Markup for edit state objects is as follows:

```
(E16,0,0,,5,1,1,0.0533333,1,45,0,0,1,0,0,0,1,5,127,7,0,0,7,0,1,1,0.0666667,0.0666667,6,6,0,0.0666667,6)
```

Value	Represents
E16	object ID (E) and version number (16)
0	z coordinate
0	flags
<i>empty field</i>	saved Lisp data (none in this example)
5	creation default fill pattern
1	creation default border width
1	gravity on (0 = off)
0.0533333	gravity radius (inches)
1	detent on (0 = off)
45	detent angle (degrees)
0	creation default dash pattern (index into array of patterns 0-9)
0	zoom off (1 = on)
1	zoom level (.25-16.0)
0	x shift (graph relocation in inches)
0	y shift (graph relocation in inches)
0	input scaling off (1 = on)
1	input scale factor
5	creation default fill color
127	creation default fill transparency (transparent; 0 = opaque)
7	creation default edge color
0	creation default edge transparency (opaque; transparent=127)
0	creation default text angle (degrees)
7	creation default text color
0	grid display off
1	grid alignment on
1	grid on top (front)
0.0666667	grid horizontal minor subdivision length
0.0666667,	grid vertical minor subdivision length
6	number of horizontal minor subdivisions per major division
6	number of vertical minor subdivisions per major division
0	rectangular grid (1 = isometric)



0.0666667	isometric grid subdivision length (inches)
6	number of isometric subdivisions per major division.

## **Charts and Images**

Charts are described in the chapter *ASCII Format for Charts*. Images are described in the chapters *ASCII Format for Image Objects* and *Binary Format for Image Objects*.

# *ASCII Format for Charts*

# 5

This chapter describes Interleaf ASCII format for charts.

In Interleaf 7, all charts are graphics objects and follow the general format for graphics object markup. All charts must occur within a *<Frame, ...>* command.

The ASCII format for charts consists of a series of records surrounded by parentheses. Each record contains fields separated by commas. The first field in each record is the record identifier. A parenthesis terminates each record, and an ASCII new-line character can follow this parenthesis.

To position the chart in its frame, graphics markup must precede all the chart records. The graphics markup starts with **c5**, which declares a chart. A comma must follow this graphics markup, and a two-digit chart version number follows the comma. The chart version number is **07**. No comma follows the version number, but a new line can.

---

## Record Structure

There are three basic record types for Interleaf ASCII chart markup: records for chart style control, data records, and records for variable style control.

Every chart record identifier begins with the letter *c* followed by one or more characters; for example, *cv*, *cO*, *clo*. Many records require multiple fields separated by commas. Except for the *v* record, you can omit any of the chart records. When you omit a record, the system assumes the default values for that record.

Within a given record, you can also omit fields. If you omit all fields except the record id, the record need not appear at all. If you omit some fields, you must represent the empty or blank fields with ASCII space or tab characters surrounded by the requisite number of commas. Omitted fields take default values. To minimize markup, Interleaf 7 generally does not write out fields whose values are the defaults.

In this chapter, hex data is preceded by *0x* for clarity, but this representation is not in the file data, which contains only the hex digits *0-f* or *0-F*.

## Units in ASCII Markup for Charts

Geometry for charts is specified in rsu's (ridiculously small units), signed decimal integers represented in 32 bits.

The internal representation of chart data contains information about the display precision of data input from files or menus. Therefore, when preparing data for the chart system, do not input numbers such as 4.999999 when 5.0 or 5 is sufficient. Conversely, this display precision, which can be of use in financial applications, is not lost. This precision has a more noticeable effect on the Chart Editor Data dialog boxes than on the display of the chart itself.

Interleaf takes no special care about the precision of floating-point calculations, which can thus be somewhat platform-dependent. This treatment has no discernible effect on the chart display, and all platforms conform to IEEE floating-point standards.

In the following sections, the term VALUE denotes a floating-point number whose display is precise; *short* denotes 16-bit signed integers; and *long* denotes 32-bit signed integers.

In the units described in the following section, rsu (scalable) specifies that the unit can be scaled and that its value in that case is the value for a 1,000,000-rsu-square chart. When you scale the chart interactively, some chart geometry scales automatically. This scaling only takes place under control of the *scaleflags* bits in the **f** record. In this case, such units are ratios to 1,000,000 rather than absolute units.

---

## ASCII Record Markup for Charts

**Format for Record Description**      The record descriptions in this chapter have the following form:

*id*      *fieldname[,fieldname, ...,fieldname]*      *units*

---

**Note**

Each record id must begin with **c**. In this chapter, for clarity, the required initial **c** is omitted from each record id.

### The **v** Record

The **v** record must not be omitted and must be the first record. These values control the amount of data read and must correspond to the amount of data presented in certain cases, as described in the following sections.

The format for the **v** record is as follows:

**v**      **n<sub>i</sub>, n<sub>d</sub>**      **short**

Each chart consists of **n<sub>d</sub>** dependent variables, which can take data for each of the **n<sub>i</sub>** independent values. Each variable is represented in a chart by a different texture, color, or line style. On the Chart Editor Data dialog box, the dependent variables appear as columns and the independent values appear as rows.

Line charts and their relatives have two numbers for each independent value. These numbers represent the **ordinate** and **abscissa** as described in this chapter. Other chart types associate only one number, which represents the actual data, with each independent value. You can think of these cases as having  $n_d$  collections of data indexed by the integers  $1..n_i$ . There is also a provision made for entering  $n_i$  labels and treating them in a way that depends on the chart type.

## Records for Chart Style Control

This section describes records that control the style of the chart.

### The **t** (Type) Record

The value of the **t** record determines the basic chart type.

<b>t</b>	<b>type</b>	<b>short</b>
----------	-------------	--------------

The supported types are

- 0 vertical bar chart
- 1 horizontal bar chart
- 2 100% bar chart
- 3 line chart
- 4 horizontal bar surface chart
- 5 vertical column surface chart
- 6 pie chart
- 7 line surface chart

If the chart type value is omitted, the system defaults to 0, a standard bar chart.

### The **f** (Flags) Record

The format of the **f** record is as follows:

<b>f</b>	<b>dflags, scaleflags, sflags</b>	<b>hex</b>
----------	-----------------------------------	------------

The default for the **f** record is 0,0,200.

Several bit fields control chart display behavior. These are 16-bit quantities specified in hexadecimal notation with digits *0-f* or *0-F*. Observe default behavior carefully; some of these bits specify that something be turned on, and others specify that something be turned off. For convenience, symbolic names are given for these bits.

The *dflags* control data calculations for the chart. They are as follows:

- RNGI 1 abscissa (independent value) range limits  
set by user
- RNGD 2 ordinate (dependent variable) range limits  
set by user

These flags are used only for line charts. If these bits are not set, the system computes the range of displayed data by a heuristic that places the maximum data point about 90 percent of the way up the chart.

The *scaleflags* control the behavior of the chart when it is resized. When the appropriate bit is set, a given quantity is interpreted as scaled rather than absolute rsu's, thereby changing its semantics from rsu's into ratio to 1,000,000. All values default to 0.

The principal application of this kind of relative scaling is to ensure that graphics objects overlay charts when sized with the chart, and that they keep their position and size relative to all the chart elements. If you permit margins to scale down too far, labels disappear.

The *scaleflags* are

DMAR 0x1 data margins

DBARBWD	0x2	bar border width
DBORD	0x10	data border
DBARWD	0x20	bar width
DGAPWD	0x40	gap width
DTXTMAR0	0x100	left label margin
DTXTMAR1	0x200	right label margin
DHMJWID	0x400	major hash width
DHMJLEN	0x800	major hash length
DHMNWID	0x1000	minor hash length
DBACKWID	0x4000	background line width
DLINWID	0x8000	line chart line width

The *sflags* for style control elements of the chart display style are

BARWDFIX	0x1	User sets the width of bars.
YMJ_NOLABEL	0x2	Do not label vertical hash marks.
XMJ_NOLABEL	0x4	Do not label horizontal hash marks.
LINEPTS	0x8	Data point markers drawn on lines.
BKDINFRONT	0x10	Background lines drawn in front of bars.
NOBARBRDR	0x20	No bar borders drawn.
YLOG	0x40	Log scale on ordinate axis.
XLOG	0x80	Log scale on abscissa axis.
DECIMAL_COMMA	0x100	Display decimals with commas instead of periods
PEN_LINES	0x200	Use penned lines instead of mitred lines in line charts. If you omit the <i>sflags</i> field, lines will be penned. However, if the <i>sflags</i> field is present, you must set the PEN_LINES bit to get penned lines.
YMN_LABEL	0x1000	Label minor hash marks in Y axis.
XMN_LABEL	0x2000	Label minor hash marks in X axis.
		Normally, these flags are only set for log charts, since in linear charts setting these flags destroys the distinction between major and minor hash marks. In log charts, the major hash marks are at constant multiples of the logarithm base (for example, 1, 10, 100, . . .), whereas the minor marks are at integral multiples of the major (2, 3, 4, . . . 9, 20, 30, . . .).
PIE_NOON	0x4000	Start pie charts at 12 o'clock instead of 3 o'clock.

## The g (Gap Ratio) Record

The format for the **g** record is as follows:

**g**            *gaprt*            **short**

This decimal short controls the **gap ratio**, the ratio of bar width to gap width. This ratio is given by  $(\text{gapwidth}/\text{barwidth}) = \text{gaprt}/100$ . The default value is 67, so that the gap between bars is about two-thirds the size of the bar width. This ratio is ignored if the BARWDFIX bit is set in *sflags*.

**The h (Hash Interval) Record**

The format for the **h** record is as follows:

```
h      Yhashintvl, Xhashintvl      float
```

The **h** record controls the major hash intervals on the ordinate and abscissa axes. If the **h** record is unspecified, the system determines them heuristically. *Xhashintvl* is ignored for all charts except line charts. For horizontally oriented charts, *Yhashintvl* is on the x-axis.

**The j Record**

The format for the **j** record is as follows:

```
j      txtmar[0],txtmar[1]      rsu(scalable)
```

The **j** record controls the distance from the label to the chart edge. This distance is the space for labels and hash marks. *Txtmar[0]* is at the left; *txtmar[1]* is at the bottom of the chart.

**The jr Record**

The format for the **jr** record is as follows:

```
j      txtmar[2]      rsu(scalable)
```

The **jr** record controls the distance from the label to the chart edge. This distance is the space for labels and hash marks. *Txtmar[2]* is at the right of the chart.

**The m (Margin) Record**

The format for the **m** record is as follows:

```
m      mleft, mright, mtop, mbottom  rsu(scalable)
```

This record controls the margin between the chart data border and the frame boundaries.

**The b (Data Border) Record**

The format for the **b** record is as follows:

```
b      databorder      rsu(scalable)
```

The **b** record controls the width of the data border. The default value is 2, which is the smallest line deemed reasonable in appearance on the target output device.

**The w (Bar Width) Record**

The format for the **w** record is as follows:

```
w      barwidth, gapwidth      rsu(scalable)
```

.....



This record controls the width of bars and the gaps between the bars in case the BARWDFIX flag is set in *sflags*. Otherwise, the widths are computed by the chart system. The actual width of bars is (*barwidth*+125,000) *rsu*; the actual width of gaps is *gapwidth* for all but surface charts. Surface charts have 0 *gapwidth*.

## The l Record

The format for the **l** record is as follows:

```
l      hmjwid, hmjlen, hmnwid, hmnlen, backwid, linwid rsu (scal  
                                           able)
```

The first four fields denote the width (*wid*) and length (*len*) of major (*mj*) and minor (*mn*) hash marks. The remaining fields are the widths of background lines and the lines for line charts.

## The lo (Logbase) Record

The format for the **lo** record is as follows:

```
lo      logbase      float
```

This record controls the base to which log charts are computed. The default is 10. Values must be greater than 1.0.

## The x (Offset) Record

The format for the **x** record is as follows:

```
x      barloffset    rsu
```

This record controls the offset before the first bar in a chart if this offset is fixed by the user (BARWDFIX set in *ctsflags*).

## The y Record

The format for the **y** record is as follows:

```
y      zerowid      rsu
```

This record controls the width of the origin line when it is visible. It defaults to *backwid*.

## The p Record

The format for the **p** record is as follows:

```
p      pierad,exprad,expshf    short
```

The first field in this record, *pierad*, is the ratio of the displayed radius to the system-determined value of the radius of pie charts. The second field, *exprad*, is the ratio of the radius of exploded wedges to the radius of the pie chart. The third field, *expshf*, is the shift of an exploded wedge expressed as a percentage of the radius of the pie. All are in integral percentages.

## The r (Range) Record

The format for the **r** record is as follows:

```
r          mnIrng,mxIrng,mnDrng,mxDrng          VALUE
```

The fields in this record are the minimum and maximum, respectively, of the *I* range and *D* range. When the chart is not automatically scaled (RNGI or RNGD bits set in *dflags*), these fields specify the ranges for abscissa and ordinates. The abscissa data is always ignored in all charts but line charts, because in all other charts the abscissa data is simply an index. If you set these fields explicitly, you must also set *hashintvl* in the **h** record.

## Data Records

This section describes the presentation of data. The first data records described are the simple cases. When data is read, the variables  $n_i$  and  $n_d$  must be defined.

The format for a data record is as follows:

```
D          label,data1,data2,...datand text,VALUE, . . . ,VALUE
```

The chart reader expects up to  $n_i$  data records, each preceded by a **D** identifier and containing a label and  $n_d$  data fields. Both the label and data can be omitted, but the requisite number of commas must be in the record.

Labels are strings of 16-bit characters from the International character set encoded as 8-bit ASCII bytes. Labels follow the ASCII file format conventions for graphics objects.

Chart labels must follow the graphics object conventions for quoting in text strings. You must use a backslash (\) to quote spaces and several other restricted characters in chart labels. You can give chart labels literally only if they contain printable ASCII characters with no embedded spaces.

.....

The data is given as *VALUES*. After  $n_i$  records are read, subsequent records are ignored. Records need not be contiguous, but they are indexed 1.. $n_i$  in the order in which they appear in the file.

Each field in the data records is assigned a color and pattern for display. These color and pattern assignments are fixed for all the records. For example, they distinguish the variable being plotted in a multi-variable chart.

If the chart is a line chart, the same format can be used for abscissa values, except if the identifier is Dx and there is no label.

The general format of data records for a line chart is as follows:

**Dx**       $xdata_1, xdata_2, \dots, xdata_{nd}$       *VALUE*, . . . , *VALUE*

If  $data_1, data_2, \dots, data_{nd}$  (resp  $xdata_1, xdata_2, \dots, xdata_{nd}$ ) are from the *i*-th D (resp Dx) record, then  $(xdata_j, data_j)$  is the *i*-th point on the *j*-th line.

## Records for Variable Style Control

This section describes records that control chart style per variable.

### The F Record

**F**       $Dflag_1, Dflag_2, \dots, DFlag_{nd}$       hex short

Use the *i*-th variable if DISPD is set in  $Dflag_i$ ; otherwise, ignore it. You can consider each variable or each record as used or unused in the chart display, but the data is kept in the chart. A set of flags determines this behavior and other presently unspecified per-variable or per-record behavior. The bits in these flags are

DISPI      0x1  
DISPD      0x2

### The G Record

**G**    **I**     $flag_1, Iflag_2, \dots, IFlag_{n_i}$  hex short

Use the *i*-th record if DISPI is set in  $Iflag_i$ ; otherwise, ignore it.

These bits are all set by default in any chart created or read by the chart system. External software need not deal with these bits unless you want some bits set and some bits not set.

## The C (Color), P (Pattern), and T (Transparency) Records

The format for the **C** record is as follows:

**C**       $c_1, c_2, \dots, c_{nd}$       **integer**

The  $c_i$  field represents the color of the  $i$ -th variable's display.

The format for the **P** record is as follows:

**P**       $p_1, p_2, \dots, p_{nd}$       **integer 0-255**

The  $p_i$  field represents the pattern of the  $i$ -th variable's display.

The format for the **T** record is as follows:

**T**       $t_1, t_2, \dots, t_{nd}$       **0 OR 255**

The  $t_i$  field represents the transparency (that is, visibility) of the  $i$ -th variable's display.

Every variable in a chart is assigned color, pattern, and visibility attributes. These attributes have the same descriptions as in the graphics system, and are documented in the chapters *ASCII Format for Text* and *ASCII Format for Graphics Objects*.

Color and pattern are palette entries composed of integers between 0 and 255. Transparency is an integer: either the default 0 (visible) or 255 (invisible). If all the display is to be visible, no visibility record need appear. Invisible data is involved in all computations but is painted transparently, so that anything underneath invisible data shows through.

Borders on bars and pies are still painted for invisible data. On black and white devices, invisible and white variables are indistinguishable if nothing is underneath them.

Invisible data is not the same as unused data. Unused data remains in the chart but is not involved in display computations.

You must specify color and pattern palette entries. If you do not specify color and pattern, palette referencing in the document will be incorrect and palette editing might remove from the palette entries that are still represented in the document.

## The L (Line Style) Record

The format for the **L** record is as follows:

**L**         $l_1, l_2, \dots, l_{nd}$         **integer** 0-255

The  $l_i$  field controls the line style (dash pattern) of the  $i$ -th line. Lines in any line chart can be presented in either mitred or penned style. Mitred lines meet in sharp corners with the data point at the interior of the mitre. They can have color and texture just as bars and pie wedges do.

Penned lines appear as though drawn with a pen, identical to the drawing mechanism in the graphics system. These lines can have the same dashed and dotted patterns available in the graphics system. By default, charts have solid penned lines, but you must set the PEN\_LINES bit in the *sflags* field, or omit the *sflags* field.

## The O Record

The format for the **O** record is as follows:

**O**         $o_1, o_2, \dots, o_{nd}$         **integers**

The  $o_i$  field is the offset in increments of one-twelfth the *barwidth* of the  $i$ -th bar. In bar charts, when the chart system computes the position of the displayed variables relative to one another, it uses an offset from an ideal center. The system computes the ideal center so that each of the  $n_i$  data points is evenly spaced. Bars are offset negatively (left) or positively (right) in one-twelfth of a bar's width from this ideal center.

Each of the  $n_i$  bars representing data for a particular  $n_d$  variable is offset by the given amount from the ideal position for that data point. If any two bars have the same relative offset, their values are summed and the bars appear stacked on one another.

## The E (Endpoint) Record

The format for the **E** record is as follows:

**E**         $e_1, e_2, \dots, e_{nd}$         **0-5**

These fields are endpoints. If the LINEPTS bit is set, markers are put on the data points in line charts. These markers are chosen from a small fixed set in the order given on the Chart Editor Style sheet.

.....

## The W Record

The format for the **W** record is as follows:

**W**             $w_1, w_2, \dots, w_{ni}$             unsigned integer

If  $w_i = w$ , the  $w$ -th variable is exploded in the  $i$ -th pie chart. Defaults are 0, meaning no explosions occur.

## Sample Chart

The following chart is taken from the *Charts* folder in the *Graphics* drawer of the *System5* cabinet. The Version is 8.0. This chart has been surrounded by a dashed box to show the actual boundaries of the chart as a graphics object, which include chart margins. Thus everything in the dashed box is part of the chart.

The ASCII markup for the sample chart follows the chart. All but the essential ASCII markup for charts has been omitted here.

The last chart record is followed by a parenthesis closing the graphics record, **c5**, that declares the chart. The chart **ch** record could have been omitted since all of its fields are empty.

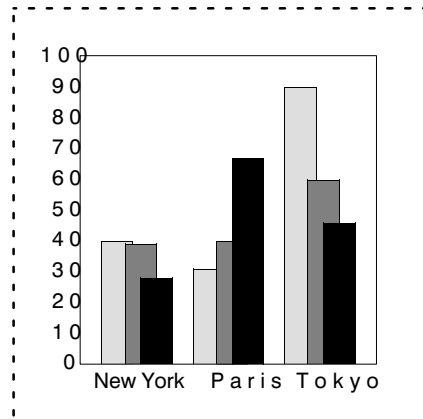
## ASCII Markup for Sample Chart

```
<|,"4<#0106>1">
<Frame,
    Name =      "figure",
    Placement = At Anchor,
    Width =     4.50 Inches,
    Height =    2.16 Inches,
    Height =    Contents,
    Diagram =
V11,
(g9,1,0,
(g9,1,0,
    (c5,1,0,,0.0533333,0.16,2.1066666,2.0456519,wst:swiss8,07
      (cv,3,3)
      (cg,67)
      (cm,360000,150000,180000,100000)
      (cj,13335,40005)
      (cl,0,40001,0,40001,0,0)
      (cx,350000)
      (cr,0,0,0,0)
      (ch,,)
      (cD,New\ York,40,39,28)
```

```
(cDx,0,0,0)
(cD,Paris,31,40,67)
(cDx,0,0,0)
(cD,Tokyo,90,60,46)
(cDx,0,0,0)
(cO,-12,-3,3)
(cC,3,5,7)
(cP,5,5,5)
(cO,-12,-3,3))
(p8,3,8,,5,5,127
(g9,3,0,
(g9,3,0,
(v7,3,65536,,0.0666667,0.0133333,2.24,0.0133333,7,0,1,5)
(v7,4,65536,,2.24,0.0133333,2.24,2.1600001,7,0,1,5)
(v7,5,65536,,2.24,2.1600001,0.0666667,2.1600001,7,0,1,5)

(v7,6,65536,,0.0666667,2.1600001,0.0666667,0.0133333,7,0,1,5))))

(E16,0,0,,5,1,0,0.0533333,1,15,0,0,1,0,-0.0133333,0,1,5,127,7,0,0,7
,0,0,1,0.066
6667,0.0666667,6,6,0,0.0666667,6))>
```



**Chart is  
inside a frame**

**c5 opens the  
chart markup**

**Chart markup  
concludes**

```
<!OPS, Version = 8.0>

[only chart markup is included]

<"para">

<|,"1">
<Frame,
    Name =                "Auto",
    Placement =            At Anchor,
    Width =                2.0666667 Inches,
    Height =               1.8933333 Inches,
    Diagram =
V8,
(g9,1,0,
(c5,1,0,,0,0,2.0533333,1.8856519,wst:swiss8,07
(cv,3,3)
(cg,67)
(cm,360000,150000,180000,100000)
(cj,13335,40005)
(cl,0,40001,0,40001,0,0)
(cx,350000)
(cr,0,0,0,0,0)
(ch,,)
(cD,New\ York,40,39,28)
(cDx,0,0,0)
(cD,Paris,31,40,67)
(cDx,0,0,0)
(cD,Tokyo,90,60,46)
(cDx,0,0,0)
(cO,-12,-3,3)
(cC,3,5,7)
(cP,5,5,5)
(cO,-12,-3,3)))>
```

---

**Note**

Like other graphics, charts have a field for saved Lisp data. In the example above, this field is empty and so appears as two consecutive commas.



# *ASCII Format for Image Objects*

# 6

This chapter describes the Interleaf ASCII format for image objects.

An image is a rectangular pattern of pixels that can be a line-art image, a continuous-tone (contone) image, or color images. The ASCII representation of the image object is similar to that of other graphics objects. It begins with a left parenthesis, contains a list of fields, and ends with a right parenthesis.

The fields can be numbers, strings, or lists of other data. They are usually separated by commas. Interleaf 7 ignores white space characters such as spaces, tabs, and new lines. These white space characters can appear at any point in the text of the image object.

---

## Image Object Markup

The following is an example of markup for a black and white image object. This markup is for the image shown in Figure 6-1. To save space, the portion of the image indicated by the ellipsis has been omitted.

```
(i18,1,0,,
  2.14,0.17,3.54,1.17,0.54,1.17,
  11,0,1102,1133,
  1,2,0.0,15,100,100,
  7,0,
  0,
  2.32,2.39,
  0,
  1,
  1103,0,0,1135,0,0,1103,1135,0,0,0,
  0,0,1103,1135,1,
  1,49008
  (X0,49008,
    07FFFFFFFFFFFFFFFFFFFFC3FFE000FE3C0000000000003FFF
    E00FFF80000000000000007E3C03FFC0000000000000007FFFF
    FFE000000000000000000000000000000000000000000000000
    0000000000000000
    ...
    3FFFFC00000000000007F780000000000000000000000000000
    00000000000000000000)
  0)
```

<b>Value</b>	<b>Represents</b>
i18	object type and version number (version 18)
1	z coordinate
0	flags field
<i>empty field</i>	saved lisp data (none in this example)
2.14, 0.17	coordinates for transformed upper-left corner of image
3.54, 1.17	coordinates for transformed upper-right corner of image
0.54, 1.17	coordinates for transformed lower-left corner of image
11, 0	coordinates for upper-left corner of cropping rectangle
1102,1134	coordinates for lower-right corner of cropping rectangle
1	contrast scales (1 = linear)
2	number of breakpoints in the contrast curve
0.0, 15	breakpoint coordinate pair
100,100	breakpoint coordinate pair
7	color
0	transparency
0	default subeditor; always 0
2.32	default width of image in inches
2.39	default height of image in inches
0	unique ID
1	serial number
1103,0,0,1135	fields for optional display rasters
0,0,1103,1135	fields for optional display rasters
0,0,0	fields for optional display rasters
0,0	coordinates of upper left corner of raster relative to progenitor image
1103,1135	coordinates of lower right corner of raster relative to progenitor image
1	depth of raster in bits per pixel
1	format of raster; a raster format of 255 is followed by the pathname to the linked image
49008	raster size in bytes (not for linked images)
(X0, ...	beginning of hex data object describing the raster (not for linked images)
...000)	end of hex data object describing the raster (not for linked images)
0)	display raster description

<b>Object ID and Version Number</b>	The first field is the object type identifier and version number, in this case i18. The <i>i</i> stands for <i>image</i> , and <i>18</i> is the current version number of the image object.
<b>z Coordinate</b>	The z coordinate is an integer that specifies the depth of the image with respect to other objects in the same graphic. Objects with higher depths can obscure the image; objects with lower depths can be obscured by the image. The sample image has a depth of 1.
<b>Flags</b>	The flags field has a bit that corresponds to each of the graphics locks. Some locks that are useful with images are <i>size</i> (bit 0), <i>rotation</i> (bit 2), and <i>aspect</i> (bit 12). The bits are numbered from least significant to most significant. Setting a bit asserts the corresponding lock.
<b>Saved Lisp Data</b>	Starting with Release 5.2, and with the V11 diagram version number, a field for ASCII Lisp method storage is included in output. This change affects all types of graphics objects. For more information, refer to <i>ASCII Lisp Method Storage for Graphics Objects</i> in the chapter <i>ASCII Format for Graphics Objects</i> .
<b>Bounding Parallelogram</b>	<p>The next six fields define the image's bounding parallelogram. These coordinate points are referred to here as <math>x_o</math>, <math>y_o</math>, <math>x_h</math>, <math>y_h</math>, <math>x_v</math>, and <math>y_v</math>. The point (<math>x_o</math>, <math>y_o</math>) is the location of the upper left corner of the original image after it has been transformed onto the page. If the image has not been rotated or reflected, this coordinate is also the upper left corner of the displayed image. The point (<math>x_h</math>, <math>y_h</math>) is the transformed upper right corner of the original image, and the point (<math>x_v</math>, <math>y_v</math>) is the transformed lower left corner.</p> <p>These coordinates are specified in inches relative to the upper left corner of the graphic. Figure 6-1 shows the sample image with the defining corners of its bounding parallelogram labeled.</p>

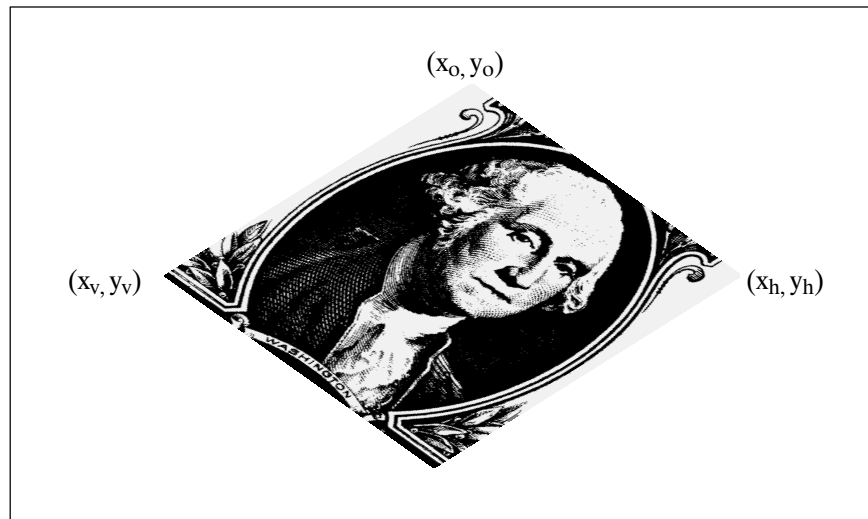


Figure 6-1. Sample image with corners of bounding parallelogram labeled.

### Cropping Rectangle

The next four fields define the cropping rectangle. The first two fields are the coordinates of the upper left corner of the cropping rectangle, and the final two are the coordinates of the lower right corner of the cropping rectangle. Both points are specified in the relation to the coordinates of the progenitor image, and the coordinates are in pixels.

Portions of the image that lie on or within the cropping rectangle appear in the document. The sample image is 1103 pixels wide by 1135 pixels high. Its cropping rectangle, therefore, indicates that 11 pixels have been trimmed from the left side and one pixel has been trimmed from the bottom. The bounding parallelogram is really the bounding parallelogram of the image after cropping, not of the entire image.

### Contrast Scales

The next field determines the scale used in the contrast editing. A value of 1 indicates linear scales, which is what all images created in Interleaf 7 use. A value of 0 indicates logarithmic scales, which were used prior to Interleaf 5. This field is used because it is not always possible to convert piecewise-linear curves based on logarithmic scales, so old documents are saved with their logarithmic scales intact. The value of this field should be 1 (linear) unless you are certain it needs to be set to 0 for backward compatibility.

## Breakpoints in the Contrast Curve

The next field defines the number of breakpoints in the contrast curve. A zero indicates that the contrast curve is the identity mapping. Otherwise, the number of breakpoints should be at least 2, and it should be followed by a list of breakpoint coordinate pairs.

The first coordinate of each pair is the abscissa of the corresponding breakpoint, and the second coordinate is the ordinate. The coordinates are expressed as floating-point percentages, with 0 representing white and 100 representing black.

The first breakpoint of the contrast curve must have an abscissa of 0. The last breakpoint must have an abscissa of 100, and abscissas of the intermediate breakpoints must increase monotonically.

In the sample image, the first breakpoint has been raised slightly to give the image a light gray background. The ordinate of this breakpoint is 15, indicating that it lies 15 percent of the way from the white end of the contrast scale to the black end.

---

### Note

In images created prior to Interleaf 5, the axes of the contrast curve are logarithmic rather than linear, and so the breakpoint coordinate values do not correspond directly to halftone dot percentages. This causes images to appear darker than the percentages in the contrast curve suggest. This is not an issue for Interleaf 7 images.

## Color

The next field defines the image color. Black portions of the original image display in the specified color when the image is part of a document. One of the values in the document's color palette should be used as the color. The color palette is defined by the `<!Color Definitions>` declaration. (Refer to the description of the `<!Color Definitions>` declaration in the chapter *ASCII Format for Text*.) Most images, like the sample, are black, which is color 7 in the default color palette.

## Transparency

The next field defines the transparency of the image. The transparency field has only two legal values, 0 and 127. A transparency of 0 means the image is visible. A transparency of 127 indicates that it is invisible.

## Default Editor

The next field identifies which of the three subeditors appears by default when you edit the image. A value of 0 corresponds to the Picture Editor. This field should always be 0.

**Width and Height**

The next two fields define in inches the default width and height of the image. The graphics feature that resets an image to its initial dimensions uses the default width and height fields. The sample image, scanned from a dollar bill, was originally 2.32 inches wide and 2.39 inches high. These dimensions refer to the entire image, not just the portion that is visible after cropping.

**Unique ID**

The next field is a unique ID intended to maintain coordination between linked images in a document and the *.img* image files to which they are linked. If an image in a document is not linked to an outside file, the value of this field is 0. For linked images, the 4-byte unique ID field of the *.img* should match the unique ID of the image instance in the document. If it does not, Interleaf 7 assumes the *.img* file has been edited and recomputes a new display resolution raster for the image when the document is opened.

**Serial Number**

The next field is a serial number (it was called a “raster identification number” in previous editions of this manual). Every distinct raster in a graphic must have a unique serial number. You can satisfy this requirement by numbering the images sequentially, starting with 1. If two or more images in the same graphic share the same raster, only the first image includes a complete raster definition; subsequent images repeat the serial number and omit the remaining fields.

Since the sample image is the only image in the graphic, its serial number is 1 and it includes a complete raster description. If you make a copy of this image with the graphics **Dup** command, the copy will have the same serial number but will not include the remainder of the raster description. This is true even if the copy has a different size, rotation, cropping, or contrast curve from the sample.

**Display Rasters**

The next 11 fields are used primarily for optional display rasters. In the main raster, the optional display rasters’ values are

```
w-1, 0, 0, h-1, 0, 0, w-1, h-1, 0, 0, 0,
```

where *w* and *h* are the width and height of the raster in pixels.

**Current Image  
Relative to  
Progenitor Area**

The next four fields define the boundaries of the current image relative to the progenitor area. The progenitor image is the original image from which the current image was derived through editing.

.....

The first two fields in this series are the x and y coordinates of the upper left corner of the current image relative to the progenitor image. The next two are the x and y coordinates of the lower right corner of the current image relative to the progenitor image. This information allows an image that is linked into a document and then later cropped externally with the Image Editor to remain in the correct position on the page. These coordinates are specified in pixels.

**Raster Depth**

The next field is raster depth, which is specified in bits per pixel. Only depths of 1 and 8 are supported. A depth of 1 is used for line-art images, and a depth of 8 is used for contone images.

**Raster Format**

The next field defines the format of the raster. Rasters can be encoded in compressed (value = 1) or uncompressed (value = 0) format, and can be local or linked in from external images. Only the uncompressed format is described here.

A value of 255 in this field indicates that the image data is contained in an external file (and linked into the document) rather than residing in the document. If this is the case, a pathname for the linked image file follows, instead of the raster size and the hexadecimal representation of raster data. Normally this is the filesystem pathname, but alternate name spaces can be implemented through Interleaf Lisp.

**Raster Size**

The next field defines the raster size in bytes. For uncompressed line-art images, this is

$$2h \left\lfloor \frac{w + 15}{16} \right\rfloor$$

For contone images, the raster size in bytes is

$$2h \left\lfloor \frac{w + 1}{2} \right\rfloor$$

where  $w$  is the pixel width of the image, and  $h$  is the pixel height.

The sample image is 49,008 bytes long. There is no comma after the size field.



## Hexadecimal Data Object

The next field is a hexadecimal data object that describes the raster itself. Like the image object, the data object consists of fields separated by commas and surrounded by a pair of balanced parentheses.

The first field is the string *X0*. The *X* stands for hexadecimal, and *0* is the version number of this object type.

The next field is the number of bytes of data followed by a string of hexadecimal digits, two for each byte of data. A right parenthesis that terminates the data object follows the last hexadecimal digit.

The data constituting the raster is interpreted in scanline-major order. That is, the data for the first scanline appears first, followed by the data for the second scanline, and so on.

For line-art images, each bit of data represents a pixel; 0 bits represent black pixels, and 1 bits represent white pixels.

For contone images, each byte represents a pixel intensity; 0 specifies black, and 255 specifies white.

For contone art, first-to-last order in the data corresponds to left-to-right order in the scanline. For line art, most significant to least significant order in the data corresponds to left-to-right order in the scanline. For efficiency, scanlines are padded to the next even byte boundary.

## Optional Display-Resolution Raster

The next field is the beginning of the description of an optional display-resolution raster. For importing an image, you can omit the display raster by supplying an identification number of 0. This was done in the sample image.

When parsing an image that contains a display raster, you can skip the display raster by reading characters until encountering the right parenthesis that terminates the image object. The display raster description is at the end of the image description.

When skipping the display raster, do not terminate prematurely. Display raster descriptions can contain right parentheses, but these right parentheses must always be balanced by matching left parentheses. Skipping should stop only when the proper right parenthesis is reached.

.....

The display raster description is followed by the right parenthesis that terminates the image object.

---

## Color Image Object Markup

The following is an example of markup for an 8-bit color image object.

(i20,	object type and version number
1	z coordinate
0,	flags field
<i>empty field</i> ,	saved Lisp date
20,	version
0,	byte count (always 0 for ASCII format), new field in I6
0,	reserved flags field, new field in I6
0.0066667,0.0066667,	position
0.8599992,0.0066667,0.0066667,1.2866658	transform
3,0,0,0,63,95,	cropping frame
3,0,1,2,0,0,100,100,	red contrast curve
3,0,1,2,0,0,100,100,	green contrast curve
3,0,1,2,0,0,100,100,	blue contrast curve
3,0,1,2,0,0,100,100,	composite (white) contrast curve
7,	color index
0,	transparency
0,	backing store ID
1,	image serial number
6,	image object version
0,	byte count
0,	flags
75,75,	resolution
0,0,	raster header offset
64,96,8,	dimensions
0,	backing store type
150994944,	null backing store format
216	number of color map entries
color map;	
(X0,648,000A285BA2FE000A285BA2FE000A285BA2FE000A285B	
A2FE000A285BA2FE000A285BA2FE000A285BA2FE000A285BA2	
FE000A285BA2FE000A285BA2FE000A285BA2FE000A285BA2F	

[illegible]

image data:

(X0,6144,5656565656565656564F56564F56565656564F5656564F565  
64F56564F564F56564F564F564F56564F564F5656565656565656565  
65656565656564F564F5656564F5656074F074F575D5C57635D5656  
07565D565D5D885D565D565D6357885D875E87885D635D5D875  
E88075608

...

0F51570957500F57575D5E565D5D56565656560E564F0E565D570  
E5D5D57815D810809075009080209090809500908515D5D5D56635  
75C5D5C575D565D5656575C575D8164568881565D5C0887575656  
5D5D56815756568888815D8188888188575051085109575851515108  
5108510788818881828757815D88815D88885D88875D818188818888  
5D88888188815D56),0)

Format Fields and Values

Notation:  
italics = definition of the object follows  
[] = optional, usually a previous field determines whether it is present  
{ } = zero or more occurrences

DG-Image Object I/O Format	Field	Comments
	version	currently 20
	byte count	
	flags	see below
	position	RSUs, transformed upper-left
	corner	
	xo	
	yo	
	transform	RSUs, relative to transformed upper-left
	upper-right	
	dxh	
	dyh	
	lower-left	
	dxv	
	dyv	
	cropping frame	uses version 3
	contrast curves	uses version 3
	R curve	var
	G curve	var
	B curve	var
	W curve	var
	color info	
	color index	index into document color palette
	transparency	0=opaque, 127=transparent
	backing store ID	
	image serial number	
	[<original image>]	var      uses version 6
	raster serial number	4
	[<display raster>]	var      uses version 5

Cropping Frame	Field	Comments
	version	currently 3
	byte count	
	upper-left	
	x1	
	y1	
	lower-right	
	x2	
	y2	

Contrast Curve	Field	Comments
	version	currently 3
	byte count	
	mapping	0=logarithmic, 1=linear
	nbreaks	
	{breakpoint}* x	
	y	

Display Raster I/O Format	Field	Comments
	version number	currently 5
	byte count	# bytes to skip, 0=unknown
	cache key	
	transformation	screen pixels
	xh	
	yh	
	xv	
	yv	
	cropping	image pixels
	x1	
	y1	
	x2	
	y2	
	contrast maps	
	flags	
	[R map]	256
	[G map]	256
	[B map]	256
	[pixel map]	256
	halftone screen	
	w	0 if no screen
	h	0 if no screen
	{thresholds}* dither	var
	type	0=none, 1=gray, 2=color
	# levels	
	<i>raster image</i>	uses version 6

Image Object I/O Format	Field	Comments
	version number	currently 6
	byte count	# bytes to skip, 0=unknown
	flags	
	resolution	
	horizontal	pixels/inch
	vertical	pixels/inch
	raster header	
	offset	
	x	
	y	
	dimensions	
	w	
	h	
	depth	
	backing store type	
	<i>backing store</i>	

---

## Backing Store Object Types

<b>Null Backing Store Object</b>	Type 0	
	<b>Field</b>	<b>Comments</b>
	format	mono/gray=0x20000000, color=0x29000000 colormap
	size	number of entries
	<i>{R data}* {G data}* {B data}* {raster data}*</i>	var var var var
<b>Compressed Backing Store Object</b>	Type 1	
	<b>Field</b>	<b>Comments</b>
	version	currently 2
	colormap size	# bytes number of entries in the colormap (0 or 256)
	size	# of bytes of compressed data
	<i>{data}*  {R data}* {G data}* {B data}*</i>	data compressed with Interleaf algorithm colormap var var var
<b>Linked Backing Store Object</b>	Type 255	
	<b>Field</b>	<b>Comments</b>
	version	currently 2
	byte count	# bytes to skip (0=unknown)
	size	# bytes in path, EXCLUDING terminator
	<i>pathname</i>	var null-terminated pathname

---

**Note** The ASCII version does not have a size field.



# *Binary Format for Image Objects*

# 7

This chapter describes the Interleaf binary format for image objects. To use the information in this chapter, you need experience working with image objects, rasters, and binary formats. You should also know how to debug image object filters, because Interleaf does not provide diagnostics or error messages for this process. If you are familiar with these topics, the information in this chapter will help you import images into an Interleaf document.

Interleaf uses the binary format for image objects to describe raster images that have not yet been pasted into a document. This binary file is separate from the ASCII markup file for a document.

---

## Image Object Binary Format

### Structure of an Interleaf Image File

A binary image file has three parts, in the following order:

- general header
- image header
- raster data

The data in the general header and in the image header is in fields of long and word size. A long field is 4 bytes, and a word field is two 2. The bytes are stored with the most significant byte first (68,000-byte ordering).

#### The General Header

The general header has two fields, shown in the following table:

Field	Value	Size
Magic number	894F5053 <sub>16</sub>	Long
Version	3	Word

---

**Note**

The value of the *version* field changes with modifications to the publishing software.

## The Image Header

The image header has nine fields, shown in the following table.

Field	Units (Value)	Size
Horizontal resolution	Pixels/inch	Word
Vertical resolution	Pixels/inch	Word
Unique ID	Number	Long
X Location	Pixels	Word
Y Location	Pixels	Word
Width	Pixels	Word
Height	Pixels	Word
Depth	Bits/pixel	Word
Format	0	Word

### Note

The value of the *unique ID* field must be a number with a high probability of not being duplicated by another image. Interleaf supports only depths of 1 bit/pixel (for line-art images), 8 bits/pixel (for greyscale or ‘pseudocolor’ contone images) and 24 bits/pixel (for color images). The value of the *format* field might vary with future releases of the publishing software; the format with a value of 0 will always be supported.

The horizontal and vertical resolutions of the image determine only the initial size of the image. For example, if you have an image that is 75 pixels high and 100 pixels wide and you set the horizontal and vertical resolution to 25, when you paste the image in a document, it will be 3 inches high and 4 inches wide. The *X Location* and *Y Location* fields describe the location of the upper left corner of the image relative to the upper left corner of the original, uncropped image.

## Raster Data

The raster data is stored as a series of *height* consecutive scan lines, starting with the top scan line of the image. Each scan line is a sequence of *width* pixels, starting with the left pixel in the scan line.

Left-to-right order on a scan line corresponds to most-significant to least-significant order within a word. New scan lines always start at a word boundary; the end of a scan line is padded on the right to the next word boundary. Pixels are packed as tightly as possible within words (16 line-art pixels or 2 contone pixels per word).

In line-art images, a pixel value of 0 represents black and a pixel value of 1 represents white. In contone images, pixels signify reflectances, with 0 representing black and 255 representing white.

## Pasting a Raster File into a Document

Once you have translated your raster data into the Interleaf format, you can paste it into a document.

1. Add the extension *.img* to the name of the raster file.
2. Move the raster file into your desktop directory.
3. Bring up your Interleaf 7desktop.

The raster file appears in the top left corner of your desktop, represented by an image icon.

When you paste an image into a document, the image is in uncompressed form. When you save the document in ASCII or binary format, Interleaf 7 saves the image in compressed form.

---

## Color Image Binary Format

Interleaf uses color image binary format for color image objects to describe color images that have not yet been pasted into a document. This binary file is separate from the ASCII markup file for a document.

## Structure of Interleaf Image Files

Interleaf image files consist of three sections, described here.

- A header
- An optional colormap
- Raster data

### Header

field	bytes
comments	
magic number	4
	0x894F5053
version	2
currently	4
resolution	
horizontal	2
pixels/inch	
vertical	
2	pixels/inch
unique id	4
offset	
x	2 pixels
y	2 pixels
dimensions	
w	2 pixels
h	2 pixels
depth	2 1, 8, or 24
type	1 0=uncompressed, 1=compressed
format	4 mono/gray=0x20000000, RGB=0x29000000
colormap size	2
number of valid colormap entries	

Fields are packed tightly together in the file. There is no padding between adjacent fields.

Multi-byte quantities are stored in big-endian byte order (the most significant byte is first in the file).

.....

The horizontal and vertical resolution fields determine the initial physical dimensions of the image when it is pasted in a document. These should be set to the resolution of the scanner that scanned the image so that the image appears actual size when pasted into a document. Alternatively, zeros can be supplied for these fields. Then the image will be scaled so that each image pixel is represented by one screen pixel.

The unique-id field should be set to a random number that has a high probability of being unique. This field is used by the image caching software to quickly determine whether the image has been modified. One possibility is to use a time stamp for the unique-id. If generating a unique-id is difficult, this field may be set to 0.

The x and y offset fields are used to keep track of how an image has been cropped. They specify the offset of the image's upper-left corner from the upper-left corner of its "progenitor" image (the original scanned image from which this image was derived). For newly scanned images, these fields should have values of 0.

The format field describes various characteristics of the raster data, such as the bit order, scanline padding, or color model. Only two formats are used: 0x20000000 for monochrome and grayscale images, and 0x29000000 for color images.

A non-zero value for the colormap size field indicates that the optional colormap is present in the file and gives the number of valid entries in the colormap. A zero value means the colormap is not present. This field should be non-zero only if the depth field is 8 and the format field is 0x29000000.

**Optional Colormap** A colormap is present only if the value of the colormap-size field in the header is non-zero. If a colormap is present, it always occupies 768 bytes, regardless of the number of valid entries it contains. The colormap is organized as follows (where  $N$  is the number of valid entries as specified by the colormap-size field of the header).

- $N$  bytes of red intensity values (without gamma correction)
- 256- $N$  bytes of padding (0 preferred)
- $N$  bytes of green intensity values (without gamma correction)
- 256- $N$  bytes of padding (0 preferred)
- $N$  bytes of blue intensity values (without gamma correction)
- 256- $N$  bytes of padding (0 preferred)

## Raster Data

Raster data is stored in scanline-major format. The data for the first scanline is first in the file, followed by the data for the second scanline, and so on. Within each scanline, the data representing the leftmost pixel comes first in the file, followed by the data for the next pixel, and so on.

In 1-plane images, pixels are represented by bits, which are packed 8 per byte. Most-to-least significant bit order within a byte corresponds to left-to-right pixel order on the scanline. A “0” bit corresponds to a black pixel, and a “1” bit corresponds to a white pixel.

In 8-plane images, each pixel is represented by a byte. If the image is grayscale, the bytes are interpreted as non-gamma-corrected intensity values, with 0 representing black and 255 representing white. If the image is a color image, the bytes are interpreted as indexes into the colormap.

In 24-plane images, each pixel is represented by 3 bytes. The bytes are interpreted as non-gamma-corrected intensity values for the red, green, and blue components of the pixel. The component values are stored in a scanline-interleaved fashion; the red components of all of the pixels on a scanline come first, followed by the green components for the scanline, followed by the blue components for the scanline. A scanline looks like the following:

```
RRRRRRRRRR . . . GGGGGGGGGG . . . BBBBBBBBBB . .
```

Each scanlines is padded to the next 2-byte boundary. For example, if the width  $W$  of an 8-plane image is odd, each scanline consists of  $W+1$  bytes. In 24-plane images, the padding is added to the end of each sub-scanline.

# Compressed Raster Format

This section describes the format used by Interleaf to represent compressed line-art images. Compressed raster format is used when an image pasted into a document is saved. It saves both disk space and memory. These images are identified in document and image files by a format field with a value of 1.

The compression format is byte-oriented; that is, a compressed image is represented by a sequence of fixed-size 8-bit units. By comparison, other compression schemes use variable-length units or units with a length other than 8 bits (for instance, 4-bit nibbles). The byte-orientation of this scheme allows it to be implemented efficiently.

A compressed image might have a structure similar to the following.

SCCCCCSCCCDCCCSDDDDDDDSCCCCCSSCCCCSCCCS

Each letter represents one byte, and the different letters represent different types of bytes. There are three basic types of byte (labeled S, C, and D in the figure). These are called *signal*, *code*, and *data* bytes. Another way of thinking of a compressed image is as a sequence of compressed scanlines. Each scanline begins with a signal byte and extends up to (but does not include) the next signal byte. The body of a scanline consists of zero or more code and data bytes. In the example, every other scanline has been underlined.

## Signal Bytes

Signal bytes mark the location of features in the compressed image. There are four signal bytes. The simplest is the EOI byte, which has a hexadecimal value of FF:

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

EOI signal byte

The EOI byte marks the end of the image. An EOI byte must appear as the last byte in the compressed image, and no EOI byte may appear at any other point in the image (although data bytes with the same value as the EOI byte may appear throughout the image). The final “S” in the sample image represents the EOI byte.



The remaining three signal bytes mark the beginning of scanlines in the compressed image. Each compressed scanline must begin with one of these signal bytes. The different signal bytes correspond to different methods of compressing the scanline. A RAW signal byte begins scanlines that, because of their complexity, could not be compressed. The hexadecimal value of a RAW byte is FA:

1	1	1	1	1	0	1	0
---	---	---	---	---	---	---	---

RAW signal byte

This type of signal byte is followed by data copied directly from the original uncompressed raster. The entire uncompressed scanline, including the final padding byte, is copied without alteration. The third scanline of the example is a raw scanline.

The remaining two signal bytes identify compressed scanlines. In both cases, the encoding method is the same; what differs is the preprocessing the scanline is subjected to before being encoded. The first of these signal bytes is the VXOR byte, whose hexadecimal value is FC:

1	1	1	1	1	1	0	0
---	---	---	---	---	---	---	---

VXOR signal byte

Scanlines beginning with this byte are exclusive-ored before being encoded. Each pixel of the scanline being compressed is exclusive-ored with the pixel immediately above it (the pixel at the same position in the preceding scanline of the uncompressed image).

In typical images, pixels of the same color tend to be clustered together. Consequently, the result of this operation is a scanline that contains mostly “0” pixels, a property that is exploited by the encoding scheme described in the next section.

The final signal byte is the HXOR byte. It too signals the start of a scanline that has been exclusive-ored prior to encoding, but in this case each pixel is exclusive-ored with its left neighbor (except for the first pixel, which is left unchanged).

This type of preprocessing is used for the first scanline of the image because it has no predecessor, but it may be used with other scanlines as well. Sometimes horizontal exclusive-oring results in better compression than vertical exclusive-oring. The hexadecimal value of the HXOR byte is FB:

1	1	1	1	1	0	1	1
---	---	---	---	---	---	---	---

HXOR signal byte

**Code Bytes**

After being preprocessed as described in this section, scanlines consist of *runs* of zero bytes that are separated by strings of non-zero *terminator* bytes. Usually there are several zero bytes per run and only one or two terminator bytes between each pair of runs. In the compressed image, these runs and intervening terminator bytes are represented by code bytes. Compression results because a run of several bytes can be represented by a single code byte.

There are three types of code byte. The RUN code byte encodes runs up to 63 bytes long. It is identified by its two most significant bits, which have values 1 and 0. The least significant 6 bits specify the number of zero bytes in the run:

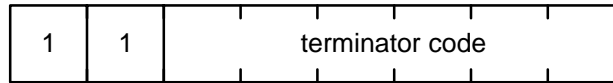
1	0						
---	---	--	--	--	--	--	--

RUN code byte

Runs longer than 63 bytes can be represented using multiple code bytes. When two or more RUN bytes appear consecutively, their run length fields are concatenated to form a larger run length. The run length fields of two RUN bytes, for example, combine to form a 12-bit run length, enough to represent runs up to 4095 bytes long. When run lengths combine in this manner, the first code byte contributes the least significant bits of the run length, and the last code byte contributes the most significant bits.

If a scanline ends with a run, the RUN bytes for that run can be omitted. A common special case is a scanline that consists of a single run. This occurs when two consecutive scanlines of the original image are identical. In this case, the description of the scanline is empty except for the initial signal byte. The fifth scanline of the sample image is an example of this.

A second type of code byte is the TERM code byte, used to encode the non-zero bytes that terminate runs. Its two most significant bits have values of 1 and 1, and its least significant 6 bits encode the value of the terminator byte:



TERM code byte

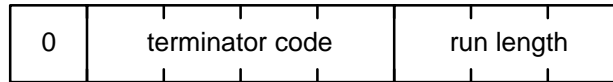
Six bits are not sufficient to encode every possible 8-bit terminator value, but they are enough to encode the most common ones. Every byte containing a single contiguous sequence of 1 bits, as well as every byte containing exactly two non-contiguous 1 bits, can be represented. The correspondence between terminator codes and terminator values is given in the following table.

Terminators that do not appear in the table are handled by placing a 0 in the terminator code field and following the TERM byte by a data byte containing the actual terminator value. For example, a terminator with a hexadecimal value of 83 is represented by the two byte sequence C0, 83. The second scanline of the sample image contains an instance of this situation.

code (hex)	value (binary) (hex)	code (hex)	value (binary) (hex)
00	<i>follows code byte</i>	1D	011111100 7C
01	00000001 01	1E	11111000 F8
02	00000010 02	1F	00111111 3F
03	00000100 04	20	01111110 7E
04	00001000 08	21	11111100 FC
05	00010000 10	22	01111111 7F
06	00100000 20	23	11111110 FE
07	01000000 40	24	11111111 FF
08	10000000 80	25	10100000 A0
09	00000011 03	26	10010000 90
0A	00000110 06	27	10001000 88
0B	00001100 0C	28	10000100 84
0C	00011000 18	29	10000010 82
0D	00110000 30	2A	10000001 81
0E	01100000 60	2B	01010000 50
0F	11000000 C0	2C	01001000 48
10	00000111 07	2D	01000100 44
11	00001110 0E	2E	01000010 42
12	00011100 1C	2F	01000001 41
13	00111000 38	30	00101000 28
14	01110000 70	31	00100100 24
15	11100000 E0	32	00100010 22
16	00001111 0F	33	00100001 21
17	00011110 1E	34	00010100 14
18	00111100 3C	35	00010010 12
19	01111000 78	36	00010001 11
1A	11110000 F0	37	00001010 0A
1B	00011111 1F	38	00001001 09
1C	00111110 3E	39	00000101 05

If a run is less than 8 bytes long and its terminator byte contains only a single 1 bit or two consecutive 1 bits, you can encode both the run and terminator with a single code byte. This is called a COMB

(for combination) code byte. Its most significant bit is 0, and it contains a 4-bit terminator code followed by a 3-bit run length:



COMB code byte

The terminator code can be one of the first 16 listed in the above table. Like the TERM code byte, the COMB code byte can have a terminator field of 0, indicating that the actual terminator value follows. COMB bytes can also combine with prior RUN bytes to represent runs longer than 8 bytes. In such a combination, the COMB byte contributes the most significant 3 bits to the run length.

## Data Bytes

Any byte that is neither a signal nor code byte is considered a data byte. A data byte can have any value, including values normally reserved for signal and code bytes; context determines whether a byte is a data byte. One example is the raw scanline described in this section. In a raw scanline, every byte except the initial signal byte is data byte.

Another example of a data byte is the byte following a TERM or a COMB byte that has a 0 in its terminator field.

## Example

Consider a small image whose first two scanlines are as follows:

```
Scanline 1:  FF FF FF C0 00 00 00 00 06 FF FF FF
Scanline 2:  FF FF FF 08 00 00 00 00 0F FF FF FF
```

To compress this image, begin with the first scanline. Since there is no previous scanline, use the horizontal method of preprocessing. Replacing each bit (except for the first) by itself exclusive-ored with its left neighbor yields:

```
80 00 00 20 00 00 00 00 05 00 00 00
```

The bits participating in the exclusive-or operation are bits from the original image. To compute the third bit, for example, you use the original second bit, which has a value of 1, rather than the preprocessed second bit, which has a value of 0.

To encode this scanline, begin with the first byte, which has a hexadecimal value of 80. The terminator code for 80 is 08, so this byte is encoded by a TERM code byte with a value of C8.

Next is a run consisting of two zero bytes terminated by a byte of 20. This combination can be represented by a COMB code byte with a value of 32.

Next is a run of 4 bytes that is terminated by a 05. A COMB byte cannot encode this terminator, so use a separate RUN byte (with a value of 84) and TERM byte (with a value F9).

The final run of 3 bytes does not appear in the encoded scanline. The complete encoded scanline (including the initial HXOR signal byte) is:

FB C8 32 84 F9

The second scanline can be vertically exclusive-ored. As before, this operation is carried out entirely with original data. The result is:

00 00 00 C8 00 00 00 00 08 00 00 00

The most compact way of encoding the initial run and terminator is to use a COMB code byte for the run and a data byte for the terminator. These have values of 03 and C8. The next run and terminator can be encoded by a single COMB byte with a value of 24, and the final run is omitted as usual. The compressed scanline (including the initial signal byte) is therefore:

FC 03 C8 24

With the final EOI signal byte appended, the size of the compressed image is 10 bytes, a factor of 2.4 smaller than the original image. This low compression ratio is due to the small size of the sample image. Generally, the compression ratio increases as the size and resolution of the original image increase. Compression ratios ranging from 10 to 20 are typical.

# 8

## *The Include Commands*

This chapter describes the `<!Include, ...>` commands. With the `<!Include, ...>` commands, you can specify the name of a separate ASCII file that you want to incorporate into your current Interleaf ASCII format file.

---

## Using the Include Commands

### Using Pathnames

Use the `<!Include, pathname>` command to insert an entire ASCII markup file into the current file. You can use the `<!Include, ...>` command to insert many different files together in one file.

Use the `<!Include Declarations, pathname>` command to insert only the declarations from an Interleaf ASCII file in the current file. The `<!Include Declarations, ...>` command takes all the declarations from a specified file and places them at the top of your current file. It incorporates component masters, frame masters, autonumber masters, and table masters into the current file.

To specify the document you want to include with `<!Include, ...>` or `<!Include Declarations, ...>`, use either an absolute pathname or a relative pathname. Relative pathnames are interpreted in relation to your desktop. If you omit the pathname and give only a filename, the desktop is the default pathname.

You can specify any pathname that is valid for your operating system. Pathnames can be up to 80 characters long. You must supply the pathname completely; wildcards are not allowed. If you use certain characters in a pathname, you must enclose the entire pathname in quotation marks. For a list of these characters, see *Quoting Conventions* in the chapter *ASCII Format Basics*.

### Nesting Include Commands

You can nest the Include commands up to 10 levels deep; your current file can contain an `<!Include, ...>` command to include a file that contains an `<!Include, ...>` command that includes another file, and so on, to four levels of inclusion.

### Using Templates

You can use the `<!Include Declarations, ...>` or the `<!Include, ...>` command to put a template of declarations in your current file. This way, you can use the template to format many separate ASCII documents.



► **To include a template file in an existing ASCII file:**

1. Using a text editor, open the ASCII document you want to mark up.

July 25, 1992  
 EDUCATIONAL FILING SYSTEMS, INC.  
 123 Alphabet Avenue  
 Westville, MA 01678  
 (413) 774-4055  
 Ernest MacIntosh

2. At the top of the file, before the first line of text (where markup normally begins), type the following:

`<!--Include Declarations, pathname of template file-->`

... or ...

`<!--Include, pathname of template file-->`

Even though the file open on your desktop might not start with the line `<!--OPS, ...-->`, Interleaf treats it as a marked-up file since the `<!--Include, ...-->` line contains the necessary `<!--` characters to signal a marked-up file.

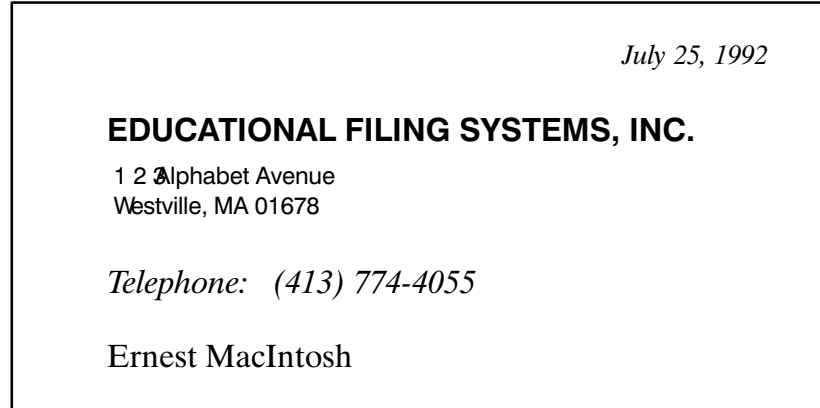
3. Your document now has the same format specified in the template file. You can use all the document components defined in the template. On the line that precedes each paragraph, section head, or other component, type the component name from the included file in command form (for example, `<date>` and `<business>`).

In Figure 8-1, the absolute pathname of the template is specified. You can also use a relative pathname, such as `address_template.doc`, or, if the template is in a folder called `ascii_temp`, `ascii_tmp.fdr/address_template.doc`.

```
<!Include Declarations, my_workstation:/u/disk1/my_login/address_template.doc>
<date>
July 25, 1992
<business>
EDUCATIONAL FILING SYSTEMS, INC.
<address>
123 Alphabet Avenue
Westville, MA 01678
<telephone>
(413) 774-4055
<space>
<name>
Ernest MacIntosh
```

Figure 8-1. Document with Include command.

4. Save the document and exit the editor.
5. If you did not create the document in your desktop directory, copy it to your desktop.
6. Open your desktop and the document.



7. When you close the document, Interleaf 7 asks if you want to save the document.

When you choose Save on the Close dialog box, Interleaf 7 saves the document as a *Fast* document; it appears on your desktop as a document icon.

If you want to edit the document outside Interleaf 7, choose Save→ASCII before you close the document. The ASCII format version of the document also appears on your desktop as a document icon, but other software programs can read the document.

On the Properties dialog box for a document or its containing directory, you can change the default Save format to Fast, ASCII, or Inherit. If you change the directory's Save format to ASCII and all the directory contents are set to Inherit (the default), Save on the Close dialog box saves these documents as ASCII documents.

When you open and then save a document that contains an Include command, Interleaf 7 combines the information from the included file and the current file into the current file. If you make subsequent changes to the included file (a template, for example), the document does not reflect these changes. The information from the included file has replaced the <!Include,...> command.

---

**Note**

You can place an include statement anywhere in markup, but the effects may be unpredictable.

## Assembling a Document

In Figure 8-2, the screen terminal displays the contents of an ASCII file called *include.doc*. When you open this file on the desktop, it becomes an Interleaf 7 document.

The template declarations and component definitions are in the file *template*. A large portion of text is in the file *part*. A previously created graphic is in the file *frame*. More text is in the file *part2*. You can assemble a complete document from these parts by using the Include command, as shown in Figure 8-2.

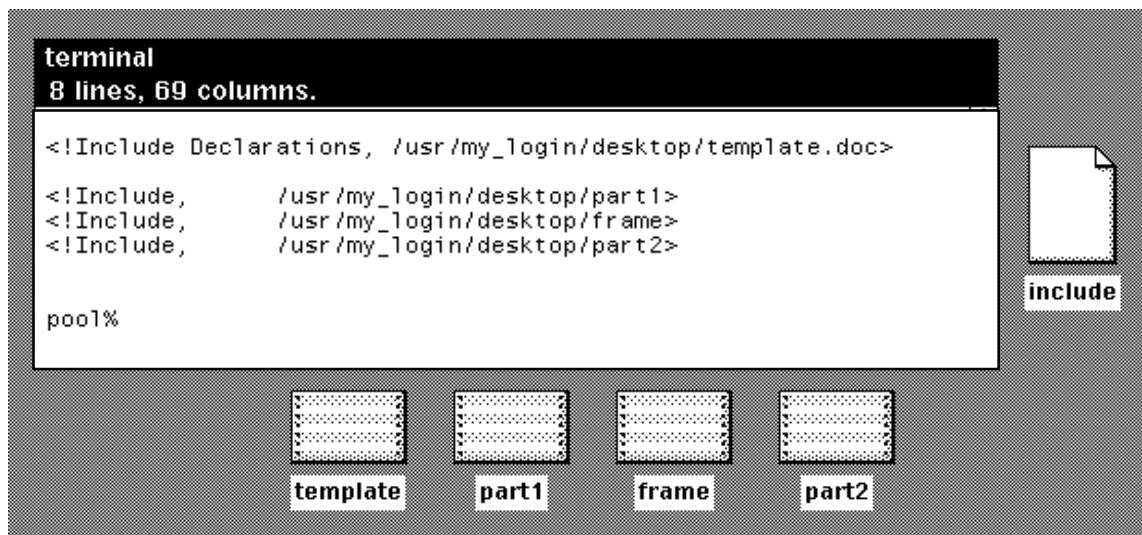


Figure 8-2. Offline document assembly.

For more complex document assembly outside the desktop, you can include many files of ASCII text and frames about a variety of topics or product parts. You can assemble these pieces in different order. This way, you can use *<!Include, ...>* commands to accomplish conditional document assembly.

You can also use database applications to assemble Interleaf 7 documents from a variety of stored ASCII data. These database applications can use stored strings of *<!Include, ...>* commands to assemble complex documents.

## Inserting Frames in an ASCII File

You can use the `<!Include, ...>` command to incorporate frames in your file. Since frames are the containers for graphics objects, they must be included if their contents are to appear. The following example shows how to insert frames in an ASCII file using the *vi* editor on the Sun UNIX platform. You can also perform this procedure using the native editor on other platforms or with the Host File Editor in Interleaf 7.

1. Create the frames in an Interleaf document and choose Save→ASCII.
2. Use the editor to put the frame definitions in separate ASCII files by typing the following:

```
% vi filename.doc <RETURN>
```

3. Use the `:se nu` command to turn on visual line numbers. The line numbers make it easier to locate blocks of text.
4. Search for the frame by typing

```
/<Frame, <RETURN>
```

This command finds the first frame in the ASCII file. The screen display resembles the following:

```
405 WP and Electronic Typewriters
406 <Frame,
407 Name = "Following Text"
408 Placement = Following Text,
409 Horizontal Alignment = Center,
```

5. Search for the end of the frame by typing the following:

```
/))> <RETURN>
```

This command puts you at the end of the frame command. Note the line numbers in the sample display:

```
564 667,6))>
565
566 <paragraph>
```

6. Save the frame as a separate ASCII file by typing

```
:406, 564w frame1 <RETURN>
```

This command creates a separate ASCII file named *frame1* in the directory or desktop in which you are working. A computer paper icon named *frame1* appears on your desktop.

7. Use the **pwd** command to check the pathname to the newly created ASCII file, *frame1*. The full pathname is required in the `<!--Include, ...>` command on the UNIX platform.

You can edit an ASCII markup file by inserting an `<!--Include,...>` statement for each file containing a frame definition. Put the `<!--Include,...>` statement for a file containing a frame definition at the point in your ASCII file where you want the frame to appear. For example:

```
<paragraph>The figure below shows the placement of the  
cursor keys.  
<!--Include, /usr/my_login/desktop/frame1>
```

The `<!--Include,...>` command does not have to appear on a line by itself.

# *Default Values for ASCII Format Documents*



If you do not provide a specific value for a given property in an input file, Interleaf automatically supplies the default value for that property. This appendix specifies the default values that the ASCII loader provides. Default settings need not be explicitly stated for input; they are automatically supplied. Non-default settings contained in input files are preserved.

The Interleaf ASCII defaults described here may not be the same as those that appear when you create a document with Interleaf 7. Refer to *Default Values* in the chapter *ASCII Format Basics* for more information.

## Default Values

### Page Number Stream

Name = "page"  
Starting Page # = Inherit  
Page # Prefix = No  
Frozen Page Numbers = Off

### Document

Header Page = On  
Double-Sided = No  
Manual Sheet Feed = yes  
Print Rev Bars = yes  
Print Strikes = yes  
Print Deletion Markers = yes  
Print Underlines = yes  
Underline at Descender = No  
Orientation Inverted = No  
Spot-Color Separation = Off  
Measurement Unit = inches  
Final Output Device = "cx"  
Default Printer = "nearest-cx"  
Line Spacing Units = points  
Font Size Units = points  
Measurement Precision = 2  
Float Precision = 2  
Points Precision = 1  
Zoom = 1  
Component Bar Width = .906 inches  
Default Page Stream Name = "page"

### Page

Orientation = Portrait  
Turned Pages = 0  
Columns = 1  
Vert. Just. = On  
Height = 11 inches  
Width = 8.5 inches  
Top Margin = 1 inch  
Bottom Margin = 1.1 inches  
Left Margin = 1.4 inches  
Right Margin = 1.4 inches  
Inner Margin = 1 inch  
Outer Margin = 1 inch  
Margins = "Left/Right"  
Vertical Margins = Add  
First Page = not set (indicates single-sided printing)  
Bleed = no  
Hyphenation = Off  
Consecutive Hyphens = Any



Allow Break After Hyphen = Yes  
Balance Columns = On  
Margin Stretch = 200%  
Margin Shrink = 50%  
Frame Margin Shrink = 10%  
Feathering = On  
Maximum Feathering = 8%  
Vert. Just. Pages = On  
Depth At Page Break = 95%  
Depth No Page Break = 90%  
Baseline to Baseline = No  
Revision Bar Placement = Auto  
Turned Pages = 0  
Frozen Autonumbers = No

**Metric page defaults:**

Top Margin = 25 mm  
Bottom Margin = 28 mm  
Left Margin = 35 mm  
Right Margin = 35 mm  
Inner Margin = 25 mm  
Outer Margin = 25 mm  
Width = 210 mm  
Height = 297 mm  
(Document Measurement Unit = mm)

**Autonumber**

Symbol Type = Arabic  
Prefix = none  
Suffix = .  
Starting Value = 1  
Last Only = No  
Show = Yes

**Component**

Name = para  
Top Margin = 0 inches  
Bottom Margin = 0.14 inches  
Left Margin = 0 inches  
Right Margin = 0 inches  
First Indent = 0 inches  
Indent Count = 1  
Begin New Column = No  
Begin New Page = No  
Composition = Overset  
Letterspacing = No  
Track Kern Spacing = No  
Line Spacing = 1.31 lines  
Alignment = Justified  
Font = Times 10



Alt Font = Times 10\*  
Hyphenation = 5 (*normal*)  
Straddle = No  
Orphan Control = 2  
Widow Control = 2  
Allow Page Break Before = Yes  
Allow Page Break Within = Yes  
Allow Page Break After = Yes  
Left Tab = 0, .75, 1.5, ... inches  
Read Only = No

## **Frame**

Name = "following Text"  
Placement = Following Text  
Height = 1 inch  
Width = 2 inches  
Horizontal Reference = Page With Both Margins  
Vertical Reference = Page With Both Margins  
Horizontal Alignment = Center  
Vertical Alignment = Bottom  
Repeating = No  
On Anchors Page = Yes  
Begin On Anchors Page = Yes  
End On Anchors Page = Yes  
Auto Edit = No  
Same Page = Yes  
Size Contents To Width = No  
Size Contents To Height = No  
Shared Contents = No  
Numbered = Off, no autonumber  
Superscript = No  
Overlap = No  
Not Selectable = No  
No Border = No

## **Table and Master Table**

A-Page = No  
Allow Page Break After = Yes  
Allow Page Break Within = Yes  
Begin New Column = No  
Begin New Page = No  
Border Visible = Yes  
Border Weight = 1  
Columns = 1  
Column N Ruling Visible = Yes  
Column N Ruling Weight = 1  
Column N Width = 1 units  
Top Margin = 0.08 inch  
Bottom Margin = 0.08 inch  
Left Margin = 0

Orphan Control = 2 rows  
Widow Control = 2 rows  
Page Break Rulings = Yes  
Straddle = No  
Right Margin = 0

## **Row and Master Row**

A-Page = No  
Allow Page Break After = Yes  
Allow Page Break Before = Yes  
Begin New Column = No  
Begin New Page = No  
Bottom Margin = 0.0266 inch  
Font = font  
Footer = No  
Header = No  
Top Margin = 0.0266 inch  
Read Only = No  
Border = No

## **Cell**

Auto Edit = Yes  
Left Ruling Visible = Yes  
Left Ruling Weight = 1  
Size Contents To Width = Yes  
Straddle = 1  
Top Ruling Visible = Yes  
Top Ruling Weight = 1  
Vertical Alignment = Top  
Not Selectable = No

## **Color**

C0 = 0, 0, 0, 0,  
C1 = 0, 0, 0, 3.125,  
C2 = 0, 0, 0, 6.25,  
C3 = 0, 0, 0, 12.5,  
C4 = 0, 0, 0, 25,  
C5 = 0, 0, 0, 50.001,  
C6 = 0, 0, 0, 75.001,  
C7 = 0, 0, 0, 100

**Pattern**

```
P0 = fdfd0000dfdfdfdfdfdf0000fdfdfdfdfd0000dfdfdfdfdf0000fdfdfdfd,
P1 = 060628286e6e8282ecec2828c0c08282060628286e6e8282ecec2828c0c08282,
P2 = 82820606080828286e6e88888080e0e082820606080828286e6e88888080e0e0,
P3 = 0c0c0c0c0c0c0c0cfcfcfcfc000000000c0c0c0c0c0c0cfcfcfc00000000,
P4 = 08080c0c0c0c1c1cfcfc78780000000008080c0c0c0c1c1cfcfc787800000000,
P5 = ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff,
P6 = c0c06060303018180c0c060603038181c0c06060303018180c0c060603038181,
P7 = 8181030306060c0c181830306060c0c08181030306060c0c181830306060c0c0,
P8 = ffffffffffffffffff0000000000000000fffffffffffffffff0000000000000000,
P9 = f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0,
P10 = f0f078783c3cle1e0f0f8787c3c3e1e1f0f078783c3cle1e0f0f8787c3c3e1e1,
P11 = 0f0fle1e3c3c7878f0f0e1e1c3c387870f0fle1e3c3c7878f0f0e1e1c3c38787>
```

\* The alternate font varies according to the value of  
ld-get-vars :japanese-default-font.

# *ASCII Format Error Messages*

# *B*

This appendix lists the error messages you might see while your ASCII document is loading, and gives the loader response to most of those errors.

ASCII Format Error Messages	
Message	Loader Response
%d Backspace characters were discarded.	Ignored; loading continues.
(Left/Right) page header/footer incorrect.	Assumes you want a Right header.
A ">" was expected but not found.	Skips to next >.
A class of that name has already been defined.	Ignores the new one.
A header or footer is redefined.	Uses the old one.
An "=" is missing.	Ignores everything until the comma or the >.
An illegal character was found.	Clears the high bit.
An undefined font is referenced.	The current font remains unchanged.
"Attach previous" is not allowed.	Ignored; loading continues.
Autonumber Stream not defined.	Uses the default autonumber.
Autonumber restart on non level one token.	Ignored; loading continues.
Bad Profile format.	Ignored; loading continues.
Cannot open include file.	Ignored; loading continues.
Chart label has been truncated to 240 characters.	Ignored; loading continues.
Chart logarithm base must be greater than 1.	
Class "... " has not been defined yet.	Creates one with the default values.
Component name longer than 9 characters.	Truncates the name.
Discretionary hyphen was ignored (n times).	Ignored; loading continues.
Distance is too big.	Truncates the distance to 36 inches.
Duplicate Master Table.	
Duplicate Master Row.	

---

**ASCII Format Error Messages (continued)**


---

<b>Message</b>	<b>Loader Response</b>
Error in Autonumber level.	If the number is too large, it is truncated to 8 and Arabic is used. If the final > is omitted, whatever has been defined up to the last level stands, and the last level takes its definition from the preceding level.
Error in frame height autosizing specification.	
Error in frame width autosizing specification.	
Error in loading a diagram.	Loads the empty frame.
Error in repeat frame.	
Error in table column number.	
Expected "Left," "Right," "Center," or "Both".	Uses the default.
Expected "Portrait" or "Landscape".	Uses "Portrait."
Expected "Yes," "No," "On," or "Off".	Uses the default.
First Indent is not consistent.	Truncated to fit. Negative indent overlapped page edge, or positive indent exceeded right margin. This error may be caused by inconsistent margin settings.
'First page' is incorrect.	Ignored; loading continues.
Font number is too high.	Assigns new number (maximum is 16,383).
Composition must be 'Optimum' or 'Overset'.	
Error in equation: Extra characters after left/right parenthesis.	
Frame has no diagram.	Creates an empty frame.
Frame not allowed in microdocument.	Discards the frame. The rest of the document might be garbled as a result.
Height is too big.	Truncates the diagram to fit.

---

ASCII Format Error Messages (continued)	
Message	Loader Response
Hex value: xx.	If it is still an illegal character, it is converted to a space.
Horizontal alignment is wrong.	Horizontal Alignment may not be specified on At Anchor frames. Otherwise, “center” is used.
Horizontal alignment not allowed with ‘At Anchor’ frame.	Ignored; loading continues.
”Hyphenation” is incorrect.	Assumes “Off.”
Incorrect placement of frame.	Ignores the placement.
Incorrect use of Straddle.	
”Include Declarations” follows some non-declaratives.	It is treated as “Include.”
Include file is not an ASCII file.	Ignores the file.
Incorrect placement of frame.	Ignored; loading continues.
Incorrect Track Kerning value.	
Invisible anchor property ignored for Numbered frame.	A numbered frame cannot have an invisible anchor. If both are specified, the invisible property is ignored.
Left/right margins are not consistent.	The component margins are set to zero. Page margins are too big if there is not at least .5 inch of room for text.
(Left/Right) page header/footer incorrect.	Assumes “Right.”
Line Spacing is out of range.	Cannot be less than the line height.
”Lines” is used incorrectly.	Uses 1/6 inch per line.
Master Diagramming object ‘%S’ was previously defined	
Master diagramming object ‘%S’ was not defined	Last instance is used for master props.
Master Frame ‘%S’ was previously defined.	The properties of the first master frame definition are used as the master for frames with this name.



---

**ASCII Format Error Messages (continued)**

---

<b>Message</b>	<b>Loader Response</b>
Master table '%S' has no valid rows to create.	Defaults are set up.
Measurement must be > 0.0.	Uses the default.
Measurement must be >= 0.0.	Uses the default.
Missing end of subcomponent.	Ignored; loading continues.
More than one diagram in frame.	Discards all except the first diagram.
More than one Master Frame with the same name.	Uses the properties of the first master frame definition for frames with this name.
Multiple tab stops at same position.	Uses the first one.
Name '=' is allowed only in Class Defaults.	Ignored; loading continues.
Negative margin goes off page.	Truncates the margin to the page margin.
No argument was found after '='.	
No master row '%S' in the table.	
No master table '%S'.	
No master frame.	Creates a default master frame. Given only for shared-content frames when the master frame is expected to have contents.
Numbered property ignored in At Anchor frame.	The numbered information is ignored.
Numbered property ignored in Repeating frame.	The numbered information is ignored.
"Page Number Type" is incorrect.	Uses "Arabic."
Page size and margins are not consistent.	Page margin values are reset to the defaults.
"Placement" must precede "Alignment", "Baseline", and "Same Page".	Uses the default.
Printer type is incorrect.	Uses the default.
Profiles on the same side may not overlap.	Ignored; loading continues.

---

.....

ASCII Format Error Messages (continued)	
Message	Loader Response
"Shared Contents" used incorrectly.	Keeps the diagram, and "Shared Contents" is turned off.
Some text was seen before any component was named.	Generates a default component.
Subcomponent property ignored in master components.	A master cannot be a submaster component. If the Subcomponent property is set to <i>Yes</i> in a master definition, it is ignored.
Starting Value incorrect.	Sets the value to 1.
Tab stops may not be set in Class Defaults.	Ignores these tabs.
Tab origin incorrect	
Superscript property ignored in non-numbered frame.	Ignored; loading continues.
The closest font has been substituted.	For example, rounds Times 9 to Times 8.
The document is not English language.	Ignored; loading continues.
The font number has been redefined.	Uses the old definition.
The format is, e.g., "FONT = F1".	Uses the default.
The format of a hex number is incorrect.	Ignores the value.
The frame is incorrect.	
The header or footer is too long.	Skips the rest, up to comma.
The hex value is not correct.	Ignores the value.
The indent count value is incorrect.	Uses the default.
The name of the font is not recognized.	Uses the <i>&lt;!Font Definitions, ... &gt;</i> default.
The number specifying the measurement is missing.	Uses the default.
The orphan value is incorrect (1 through 15).	Uses the default.
The point size is way out of range.	Assumes 10 point.

---

**ASCII Format Error Messages (continued)**


---

<b>Message</b>	<b>Loader Response</b>
The profile line count must be between 1 and 32767.	Ignored; loading continues.
The profile start line must be between 1 and 32767.	Ignored; loading continues.
The property is not allowed in a microdocument.	
The string of characters seen is too long to be legal.	Uses the truncated string; often causes further errors.
The tab measurement is incorrect.	Ignores the whole line.
The unit of measurement is unknown (e.g., 1.5 inches).	Uses the default.
The widow value is incorrect (1 through 15).	Uses the default.
The word "... " is incorrect.	Ignores everything until the next comma or >.
There are no row names specified in the table.	
There are too few cells in the table row.	Fills out the row.
There are too many cells for the number of columns in the table.	Starts a new row.
There is no declaration named "... ".	Passes the string into the document.
This declaration must precede all components.	Results are unpredictable.
This software version expects ASCII Format Ver. 7.X	
Too many </F>.	Nothing happens; the font remains the same.
Too many tokens in document.	Discards the object associated with the token.
Too many tab stops.	Ignores the extras (maximum is 30).
Top/bottom margins are too big.	Accepts both margins.
Underlining (backspace/underscore) was removed.	
Unbalanced parentheses in diagram.	Loads diagram, but in error.
Unexpected character '%c'.	

---

ASCII Format Error Messages (continued)	
Message	Loader Response
Unrecognized Horizontal Reference.	
Unrecognized Vertical Reference.	
Unrecognized component content keyword, assuming Private.	
Unknown Autonumber Symbol Type.	Uses “Arabic.”
Valid wordspace/Letterspace values are 0 through 63.99	
Value out of range	
Use “Fn” with n = 1 or more.	Uses the default.
‘Vertical Alignment’ not allowed unless ‘At Anchor’ frame.	Ignored; loading continues.
”Vertical Alignment” is wrong.	Uses “Top.”

# *Hexadecimal Codes*



This appendix lists the International character set hexadecimal codes used in Interleaf ASCII markup. For markup of special characters, see the chapter *ASCII Format for Text*.

International character set hex codes for PostScript fonts

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
<b>000 0</b>																	<b>0</b>
<b>016 1</b>																	<b>1</b>
<b>032 2</b>		!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/	<b>2</b>
<b>048 3</b>	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	<b>3</b>
<b>064 4</b>	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	<b>4</b>
<b>080 5</b>	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	<b>5</b>
<b>096 6</b>	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	<b>6</b>
<b>112 7</b>	p	q	r	s	t	u	v	w	x	y	z	{		}	~	"	<b>7</b>
<b>128 8</b>																	<b>8</b>
<b>144 9</b>																	<b>9</b>
<b>160 a</b>		ì	¢	£	¤	¥	¦	§	"	©	ª	«	¬		®	-	<b>a</b>
<b>176 b</b>	°	±				µ	¶	·			°	»	¼	½	¾	¿	<b>b</b>
<b>192 c</b>	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	<b>c</b>
<b>208 d</b>	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	<b>d</b>
<b>224 e</b>	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	<b>e</b>
<b>240 f</b>	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ	<b>f</b>
<b>256 10</b>							-	-	-		—						<b>10</b>
<b>272 11</b>					†	‡	‹	›	„							‰	<b>11</b>
<b>288 12</b>	/																<b>12</b>
<b>304 13</b>	f		lj	L·		Œ	ij	l·		œ		fi	fl				<b>13</b>
<b>320 14</b>	Ā	ā	Ă	ă	Ą	ą	Ć	ć	Ĉ	ĉ	Č	č	Č	č	Ď	d'	<b>14</b>
<b>336 15</b>	Ď	ď	Ě	ě	Ě	ě	Ě	ě	Ě	ě	Ě	ě	Ě	ě	Ě	ě	<b>15</b>
<b>352 16</b>	Ĝ	ĝ	Ĝ	ĝ	Ĥ	ĥ	Ĥ	ĥ	Ĭ	ĭ	Ĭ	ĭ	Ĭ	ĭ	Ĭ	ĭ	<b>16</b>
<b>368 17</b>	Ī	ī			Ĵ	ĵ	Ķ	ķ	κ	Ĺ	ĺ	Ł	ł	Ł	ł		<b>17</b>
<b>384 18</b>		Ł	ł	Ń	ń	Ń	ń	Ń	ń	Ń	ń	Ń	ń	Ń	ń	Ń	<b>18</b>
<b>400 19</b>	Œ	œ			Ŕ	ŕ	Ŕ	ŕ	Ŕ	ŕ	Ŕ	ŕ	Ŕ	ŕ	Ŕ	ŕ	<b>19</b>
<b>416 1a</b>	Š	š	Ţ	ţ	Ť	ť	Ŧ	ŧ	Ũ	ũ	Ũ	ũ	Ũ	ũ	Ũ	ũ	<b>1a</b>

432 1b	Ů	ů	Ů	ů	Ů	ů	Ů	ů	Ů	ů	Ů	ů	Ů	ů	Ů	ů	1b
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	

.....

## *Filter Writing*

# *D*

This appendix describes aspects of Interleaf ASCII markup procedure that may be of special interest to filter writers, such as the differences between input and output markup requirements. Major topics are listed in alphabetical order.

### **Autonumbers**

Autonumbers are automatically generated by Interleaf. Autonumber values supplied by other means cannot be guaranteed to behave properly since they may produce conflicts with automatically generated ones. Therefore, you should not determine autonumbers manually for input.

### **Pasting Autonumbers Into a Book**

If a document containing autonumbers is saved in ASCII and pasted into a book, the document must be resaved before the ASCII markup accurately reflects the new autonumber values. Until it is resaved, the document displays correct values but the ASCII markup contains the old values.



## Components (<!Class, ...> objects)

Default values for properties of component objects can be determined several ways. For more information, refer to *Default Values* in this appendix and *<!Class Defaults,...> Declaration* in the chapter *ASCII Format for Text*.

## Default Values

The procedure for assigning default values is different for component objects and frame objects. The differences are outlined and discussed in detail in the chapter *ASCII Format for Text*.

## Components

Default values for component objects are assigned the following way:

**<!Class, ...> Declaration (component).** When a *<!Class, ...>* declaration is encountered by the Interleaf ASCII loader, a master component is created with initial properties taken from a set of default values. These default values are listed in Appendix A, *Default Values for ASCII Format Documents*, subject to modification by the optional *<!Class Defaults, ...>* declaration. Any value explicitly specified for a property in that *<!Class, ...>* declaration will override the default on the master.

**<component, ...> Command.** When the Interleaf ASCII loader finds a *<component, ...>* command, a component is created with initial properties taken from the master component specified in a previous *<!Class, ...>* declaration for the given component name. Any value explicitly specified for a property in the *<component, ...>* command will override the master's value on the new component.

## Frames

With respect to default values, frames do not behave like components.

When the Interleaf ASCII loader finds a *<Frame, ...>* command, a frame is created whose initial properties are listed in Appendix A, *Default Values for ASCII Format Documents*. Initial values are not taken from the master frame, as is the case with components. Following this, the loader changes any properties specified in the *<Frame, ...>* markup that diverge from the defaults.

## Diagramming Markup

Comments are not allowed inside diagramming markup, with the exception of microdocuments.

Markup for all diagramming objects must occur inside the markup of a containing frame.

The new saved Lisp data field in diagramming markup is the fourth field. When empty, it appears as two consecutive commas after the third field in the markup for most objects; for groups it is indicated by a comma after the third field, before the opening parenthesis for the first object in the group.

## End Commands

End commands are required at the end of markup for the following Interleaf objects:

- Tables (use the *<End Table>* command)
- Microdocuments (use the *<End Text>* command)
- Inline components (use the *<End Sub>* command)

## Fill Property

On input, when *Fill = off*, hard returns are inserted at the end of each line. This is especially useful for loading text containing tabular material.

When *Fill = blank*, a blank is inserted at the end of each input line if there is not already a blank there and if it is not the last line of a component. This property is useful for treating files that have been prepared with word processors, or with editors that do not ensure blanks at line endings. When line breaks on the word processor differ from those in Interleaf 7, using *blank* ensures that words do not run together. Use this property in a *<!Class Defaults, ...>* declaration if you want it to apply to all component classes.

## Font Tokens and Font Inheritance

### Font Tokens

A font token must be the first item contained in the contents of an inline component, and a font token must appear after each *<End Sub>* marker.

When you save a document, Interleaf assigns font numbers on the basis of the fonts available at a particular time in a particular installation. Do not assume that the correspondence between font numbers and actual fonts remains constant across versions of Interleaf. If fonts are added or removed, the numbers for some or all fonts can change.

Assign your own numbers in the font declarations and maintain them consistently when you mark up documents. The software converts your numbers to the corresponding fonts when a document is loaded.

If you write a program that translates Interleaf ASCII format to another format, your program must interpret the *<!Font Definitions, ...>* declaration so that, for example, the command *<F3>* (or other *F<sub>n</sub>*) is properly translated as italic.

*</F>* command — for input only. Use this command to revert to the previous font of a component. It cannot be used to reference a font in a separate component. This command may also be used to terminate a *<FI>* or a *<FB>* command.

*<FBI>* command — for input only. This command adds bold and italic to the active font. Use the *</F>* command to terminate the command.

*<FI>* command — for input only. This command adds italic to the active font. Use the *</F>* command to terminate the command.

*<FB>* command — for input only. This command adds boldface to the active font. Use the *</F>* command to terminate the command.

### Font Inheritance

Font inheritance applies to inline components only. Through font inheritance, you can set relative inheritance for many properties, including text properties. This markup can be complex. For more information, refer to *Inline Components* in the chapter *ASCII Format for Text*.

`<F0>` command. Markup for text properties cannot be added to a default `<F0>` font token. For example, it is permissible to write `<F8@property...>` but not `<F0@property>`.

## Frames

All frame width and height values are expressed in output in addition to their margin specifications, if any. See the chapter *ASCII Format for Text* for details on other ways of marking up width and height for frames.

Table cells that contain graphic objects are a special case of frames and have special properties. Refer to the chapter *ASCII Format for Tables* for details.

## Informational Markup

The following items appear in output files but are ignored by the loader. Their values are recomputed on the fly as the document is loaded. The values as output in ASCII are for the information and convenience of filter writers.

- Autonumber values
- Line breaks (in markup: `<SR>`)
- Page breaks (in markup: `<|, ...>`)
- Page numbers on floating frames
- Edit State markup (refer to the chapter *ASCII Format for Graphics Objects*)
- Absolute frame width or height values are output when the specified size is relative to page, column size, or to the size of the contents.
- Width of soft-state documents
- *Hidden* property

## Other Information in ASCII Files

The `<!End Declarations>` declaration serves only as a convenience to delineate the end of the declarations and the beginning of the document proper. It is ignored by the loader but is useful if you are writing a filter that wants to skip over all declarations.

.....

Object names are always quoted on output. This assures consistency of naming style and eliminates any ambiguity between commands and object names. This fact can be used in filters designed to read documents generated by Interleaf 7.

## Input Markup

The following items are convenience commands that allow you to insert markup in an ASCII file that is then read and interpreted by the ASCII loader. The dumper never outputs these commands.

- `</F>`, `<FB>`, and similar commands. Refer to *Font Tokens and Font Inheritance* in this appendix.
- Convert-to-Outline object (converts text string into outline font characters). Refer to the chapter *ASCII Format for Graphics Objects* for details.

## Interleaf ASCII Method Storage

For releases of Interleaf 5 with an ASCII version number 8 and higher, Lisp expressions may be attached on output. This affects both diagramming (graphics objects) and text. Refer to the chapters *ASCII Format for Text* and *ASCII Format for Graphics Objects* for more information.

## Microdocuments

Properties for microdocuments are similar but not identical to properties declared under the `<!Page, ...> Declaration`. Refer to `<!Page, ...> Declaration` in the chapter *ASCII Format for Text* and *Microdocuments* in the chapter *ASCII Format for Graphics Objects* for more information.

## Names

When naming objects (components, subcomponents, or frames, for example) for input, the following procedures must be followed:

- Names may not exceed 19 characters.

- Names must not be duplicated for the same object type. For example, no two master components (classes) can have the same name. A frame and a component, however, may share the same name.
- Component names may not clash with reserved Interleaf names such as *cell*. A component whose name is on the following list of reserved names, or a component that uses a name reserved for font markup (for example, *F0*), must have its name in quotes. In general, we recommend that all component names be enclosed in quotes; this is always permitted, and component names are quoted on output in any case.

Autonum	LeftPageHeader
Cell	Mail
Comment	Master Row
Data	Note
EndInline	PageFooter
EndSub	PageHeader
EndTable	Query
EndText	Ref
FirstLeftPageFooter	Rev
FirstLeftPageHeader	RightPageFooter
FirstPageFooter	RightPageHeader
FirstPageHeader	Row
FirstRightPageFooter	SP
FirstRightPageHeader	SR
FJ	Tab
Frame	Tab-
HR	Tab-dash
Index	Tab.
Language	Table
LeftPageFooter	Tab_

## Newlines and Broken Words

In Interleaf 7, the system has a maximum output line-length of 79 characters. Anytime a text character is output to an ASCII format file so that its column position would exceed that value, a new line is started, sometimes causing a word to be broken across two lines (hyphenation).

The ASCII dumper tries to enhance readability by starting a new line anytime it outputs a character immediately following a space when the output column count is over 66. Also, newlines may be started before processing markup for certain tokens such as autonumbers and font changes.

In the case of font changes, when the output column count is greater than 72, a newline is generated prior to output of the font markup. In the customer sample (*94083968\_first.doc*), a font change occurs at column 64—below the newline threshold. The next text character after the font markup is in column 75, but since the previous character is not a space, the newline is not triggered until column 79 is reached.

The results are different in *94083868\_two.doc* because the component is the first on the page, so page-break markup is included in the line. This changes line count so that the maximum count of 79 has already been exceeded when the first character after the font markup is to be output. Hence a new line is started.

By putting a space after a font change instead of before it, you can prevent some words being broken, but there is no way to guarantee that words do not get broken.

Newlines may legally occur almost anywhere. A single newline character in a marked-up ASCII format file is not interpreted as having any special significance when read by the ASCII loader. Any post-processing of the output should ignore single newlines.

A forced line break is never preceded by a space (0x20)—that is, the combination 0x20 0x0a may be treated as a soft return. 0x0a 0x0a is end-of-component (or other delimiter), but 0x0a by itself may be effectively ignored.

---

**Note**

In Interleaf 6.1, you can use Lisp to control line-break values by setting the maximum line length to a high value.

## Quotation Procedures

Backslashes, not double quotes, must be used in graphics and chart markup to preserve spaces between words.

# Index

*Italicized numbers refer to pages on which figures appear.*

## Symbols

- , (comma)
  - as property and value separator, 1-7
  - in graphics objects markup, 4-2
- @~attr, convention for negating (toggling) text inheritance, 2-47
- @i\*, convention for inline inheritance (all text properties), 2-47
- @iproperty, convention for inherited text property, 2-47
- @X, caps and small caps (text property), 2-50
- +, inherit and increase operator, 2-47
- , inherit and decrease operator, 2-47
- <...>, convention for commands, 1-4
- <#xx> command, for special characters, 2-60
- <|, ...> (page break command), 2-57–2-63
- ~, inherit and toggle operator, 2-47

## A

- a (arc), in arc (graphics objects) markup, 4-14
- A4, defaults for, 2-2
- Accented Characters, in Interleaf ASCII markup, 1-8
- Arcs, graphics objects markup for, 4-12–4-32
- ASCII Dumper, 1-2
- ASCII Format
  - See also* Interleaf ASCII Format
  - for files without markup, 1-11–1-14
- ASCII Loader, 1-2, 1-13

- Attributes, user-defined, 2-61
- <Autonum, ...> Command, definition of, 2-39–2-63
- <!Autonumber Stream, ...> Declaration
  - default values for, A-3
  - definition of, 2-18–2-63
  - order in declaration, 1-5
- Autoreference, 2-40–2-63

## B

- b Record, in chart markup, 5-7
- Bézier Objects, in graphics objects markup, 4-27
- Binary Image
  - file structure of, 7-2, 7-4
  - raster data for, 7-4
- Bitmap, markup for patterns, 2-15
- Blanks
  - in Interleaf ASCII markup, 1-11
  - multiple blank lines, 1-11
- Brackets, conventions for using in ASCII markup, 1-3–1-14

## C

- C Record, in chart markup, 5-11
- Caret Position, Save ASCII commands and, 1-14
- <Cell, ...> Command
  - default values for, A-5
  - definition of, 3-9–3-18
- Character Mapping, in Interleaf ASCII markup, 1-13–1-14
- Character Translation Table, 1-11



## Charts

- in Interleaf ASCII markup, 5-1
- individual records for, 5-3–5-15
- record structure of, 5-2–5-15
- sample ASCII markup for, 5-13–5-15
- unit of markup used, 5-2–5-15

## <!Class Defaults, ...> Declaration

- definition of, 2-19–2-63
- order in declaration, 1-6

## <!Class, ...> Declaration

- component, default values for, A-3
- duplication of names as error, 2-20
- order in declaration, 1-6
- properties of, component definition, 2-20–2-63

## CMYK Color Model, 2-13

## Color and Pattern Palettes, general format for, 2-13–2-63

## <!Color Definitions, ...> Declaration

- code numbers for, 2-14
- default values for, A-5
- definition of, 2-13–2-63
- order in declaration, 1-6

## Commands

- definition for markup, 1-4
- Force Justify, 2-53–2-63
- Interleaf ASCII commands, 2-39–2-63

## <!Comment, ...> Declaration, definition of, 2-38–2-63

## Comments

- not allowed in diagramming data, 1-7
- within markup brackets, 1-7

## <component, ...> Command, definition of, 2-42–2-63

## Components

- declaration for (Class), 2-20
- master definitions for, 1-3

## Conditional Content, ASCII markup for, 2-61–2-63

## Contone Image, in Image markup, 6-1

## Convert □to□Outline Object, graphics objects markup for, 4-18–4-32

## D

### Dashes, within comment, 1-7

### Data Records, in chart markup, 5-9

### Declarations, definition for markup, 1-3

### Default Values

- document, A-2
- for Interleaf ASCII markup, A-2
- page, A-2

### dflags, in chart markup, 5-5

## <!Document, ...> Declaration

- definition of, 2-4–2-63
- order in declaration, 1-5

### Documents, creating in ASCII format, 1-14

### Dumper, 1-2

## E

### e (ellipsis), in ellipsis (graphics objects) markup, 4-11

### E Record, in chart markup, 5-12

### Edge and Fill Properties, in graphics objects markup, 4-6

### Ellipses, graphics objects markup for, 4-11–4-32

### Encapsulated PostScript Objects, markup for, 4-23–4-32

## <!End Declarations> Declaration

- definition of, 2-38–2-63
- order in declaration, 1-6

## <End Inline>, for inline components, 2-45

## <End Sub>, for inline components, 2-45

## <End Table> Command, in table markup, 3-11

## <End Text> Command, in microdocument markup, 4-20

### EPS Objects, graphics objects markup for, 4-23

### Error Messages, 1-13

- for diagrams, 4-6–4-32

## F

- F Record, in chart markup, 5-10
- f Record, in chart markup, 5-4
- </F>, font command for reverting within component, 2-52
- <FB>, font command for bold, 2-53
- <FBI>, font command for bold italic, 2-53
- <FI>, font command for italic, 2-53
- Field Specifications, flags, 4-4
- File
  - ASCII file without markup, 1-11
  - in ASCII format, 1-14
- File Type, extensions for files in, 1-12–1-14
- <FJ>, force justify command for text, 2-53–2-63
- Flags Field, in graphics objects markup, 4-3, 4-4–4-32
- <Fn>, command for font change, 2-49
- <!Font Definitions, ...> Declaration
  - definition of, 2-11–2-63
  - font size limits, 2-12
  - order in declaration, 1-5
  - procedure for font numbers, font numbers, D-4
- Font Inheritance, in inline components, 2-46
- Font Token, in inline component markup, 2-46
- Fonts, new features for, D-6
- Fonts and Font Attributes Commands, 2-49–2-63
- Foreign Language Codes, for spelling and hyphenation, 2-51
- <Frame, ...> Command, 2-54–2-63
  - default values for, A-4
- Frozen Composition, and Save ASCII commands, 1-14

## G

- g (group), in group (graphics object) markup, 4-9
- G Record, in chart markup, 5-10
- g Record, in chart markup, 5-6

## Graphics Objects

- definition of, 4-1
- general format for markup, 4-2–4-32
- layering within a frame, 4-4–4-32
- object types, letters and version numbers for, 4-3–4-32

Groups, graphics objects markup for, 4-9–4-32

## H

- h Record, in chart markup, 5-7
- Hard Return, 1-11
- Hexadecimal Data Object, in Image markup, 6-9
- Hidden = Yes, No, property in markup, 2-54
- <HR>, command for hard return, 2-55
- Hyphenation Points, Save ASCII commands and, 1-14

## I

- Icon, use in file type, 1-12
- Identifiers, for graphics objects, 4-3
- Images
  - binary format for
    - See also* Binary Image
    - caution when working with, 7-1
    - overview, 6-1–6-16
    - pasting raster images into Interleaf documents, 7-4–7-14
    - raster data for, 6-7
    - sample Interleaf ASCII markup for, 6-1–6-16
- <!Include Declarations, ...> Command, 2-55–2-63
  - overview, 8-2
- <!Include, ...> Commands, 2-55–2-63
  - assembling a document with, 8-6–8-8
  - inserting frames with, 8-7–8-8
  - nested, permissible level of nesting in, 8-2
  - overview, 8-2
  - supplying a template with, 8-2–8-8
  - using pathnames with, 8-2
- <Index, ...> Command, for index tokens, 2-56–2-63
- Inheritance, for text and font properties, 2-46

Inline Component Command, subcomponent,  
2-45–2-63

## Input

A-page markup unstable for input, 2-44  
caution with profile settings, 2-27  
Convert-to-Outline object, only for input, 4-18  
</F> Command, for input only, 2-52  
Hidden = Yes, No ignored by loader, 2-44, 2-55,  
3-8  
page and component margins, 2-8  
page break commands, no effect on input, 2-58  
page number values do not affect, 2-54

## Interleaf ASCII Format

creating documents in, 1-14  
definition of, iii

## Interleaf Binary Format

definition of, iii  
for image objects, 7-1–7-14

International Characters, in Interleaf ASCII markup,  
1-8

## J

j Record, in chart markup, 5-7

jr record, in chart markup, 5-7

## K

Knots, in markup for splines (graphics objects), 4-15

## L

L Record, in chart markup, 5-12

l record, in chart markup, 5-8

Language Codes, for spelling and hyphenation, 2-51

Line-Art Image, in Image markup, 6-1

Lines, graphics objects markup for, 4-8–4-32

Linked Images, in image markup, 6-8

lo Record, in chart markup, 5-8

Loader, 1-2

Locks, in graphics objects markup, 4-4

## M

m Record, in chart markup, 5-7

Master Definitions, for components, 1-3

<!Master Diagramming Object, ...> Declaration,  
order in declaration, 1-6

<!Master Frame, ...> Declaration  
definition of, 2-31–2-63  
order in declaration, 1-6

<!Master Row, ...> Declaration  
default values for, A-5  
definition of, 3-7–3-18

<!Master Table, ...> Declaration  
default values for, A-4  
definition of, 3-2–3-18  
order in declaration, 1-6

Measurement, units of, in Interleaf ASCII markup,  
1-10

Metric, metric options, 2-2

Microdocuments, markup for, 4-19–4-32

## N

n (EPS object), in Encapsulated PostScript Object  
(graphics objects) markup, 4-23

N (named graphics objects), in named graphics ob-  
jects markup, 4-28

## Named Graphics Objects

as new feature, D-3  
in graphics objects markup, 4-28

Newlines, Fill property and, 2-19

No Border, 2-55

## O

o (Convert-to-Outline object), in Convert-to-Outline  
object (graphics objects) markup, 4-19

O Record, in chart markup, 5-12

Object Type, in graphics objects markup, 4-2

Object Version Number, in graphics objects markup,  
4-2

OLE Objects, in graphics objects markup, 4-25

<!OPS, ...> Declaration  
 definition of, 2-2–2-63  
 order in declaration, 1-5

Output  
 importance of Hidden property, 2-43  
 importance of translating font markup, D-4

## P

p (poly), in poly (graphics objects) markup, 4-10

P Record, in chart markup, 5-11

p Record, in chart markup, 5-8

<Page Header, ...> Command, definition of,  
 2-58–2-63

<!Page Number Stream, ...> Declaration  
 as new feature, D-3  
 default values for, A-2  
 definition of, 2-2–2-63  
 order in declaration, 1-5

<!Page, ...> Declaration  
 definition of, 2-7–2-63  
 new features in, D-8  
 order in declaration, 1-5

Pattern Definitions, declaration for, 2-15–2-63

<!Pattern Definitions, ...> Declaration  
 code numbers for, 2-15  
 definition of, 2-15–2-63

Plotter Objects, markup for, 4-21–4-32

Poly, in graphics objects markup, 4-10–4-32

Prefix, markup for, 2-28

Progenitor Area, in image markup, 6-7

Property  
 conventions for values of, iv  
 default values for, overview, 1-4

## Q

Quotation Marks, in Interleaf ASCII markup, 1-8

Quoting, conventions for, 1-8–1-14

## R

r Record, in chart markup, 5-9

Raster Format, and linked images, 6-8

<Ref, ...> command, definition of, 2-40–2-63

<!Revision Tracking, ...> Declaration  
 definition of, 2-17–2-63  
 order in declaration, 1-6

<Row, ...> Command  
 default values for, A-5  
 definition of, 3-8–3-18

rsu (ridiculously small unit), definition of, 5-2

## S

S (spline), in spline (graphics objects) markup, 4-16

Save ASCII Command, hyphenation changes and,  
 1-14

scaleflags, in chart markup, 5-3, 5-5

Serial Number, in image markup, 6-7

sflags, in chart markup, 5-6

<SP>, command for hard space, 2-59–2-63

Spaces, markup for, 2-59

Special Characters, 1-8  
 command for representing in markup, 2-60–2-63  
 in Interleaf ASCII markup, 1-8

Specifications for Interleaf ASCII Format, com-  
 mands, 2-39–2-63

Splines, graphics objects markup for, 4-15–4-32

<SR>, command for soft return, 2-55

## T

T (microdocument), in microdocument (graphics ob-  
 jects) markup, 4-20

t (text string), in text string (graphics objects) mark-  
 up, 4-17

T Record, in chart markup, 5-11

t Record, in chart markup, 5-4

<Tab, ...> Command, 2-59–2-63

<Table, ...> Command

default values for, A-4

definition of, 3-8–3-18

Tables, Interleaf ASCII markup for, sample of,  
3-11–3-18

Templates, supplying with Include commands,  
8-2–8-8

Text Strings

markup for, 4-17–4-32

quoting characters in, 4-18

## **U**

Units of Measurement, in Interleaf ASCII markup,  
1-10–1-14

User-Defined Attributes

each occurrence must be specified, 2-20

use in document markup, 2-61

## **V**

V (plotter or vector-list object), in plotter (graphics  
objects) markup, 4-21

v (vector), in line (graphics objects) markup, 4-8

v Record, in chart markup, 5-3

Vector□List Objects, markup for, 4-21–4-32

Version Numbers, for graphics objects, 4-3

Visibility, field for graphics objects, 4-6

## **W**

W Record, in chart markup, 5-13

w Record, in chart markup, 5-7

## **X**

x Record, in chart markup, 5-8

## **Y**

y Record, in chart markup, 5-8

## **Z**

z (Bézier object), in Bézier object (graphics objects)  
markup, 4-27

z coordinate, in graphics objects, 4-2

# Properties

## Symbols

@A, all small caps (text property), 2-50  
 @attribute, under Row cmd, 3-8  
 @attribute = value, under Cell cmd, 3-9  
 @B, subscript (text property), 2-50  
 @C, all caps, 2-50  
 @D, double underbar (text property), 2-50  
 @K, track kerning (text property), 2-50  
 @L, language for spelling/hyphenation (text property), 2-50  
 @O, overbar (text property), 2-50  
 @P, pair kerning off (text property), 2-50  
 @R, revision bars (text property), 2-50  
 @S, strikethrough (text property), 2-50  
 @T, superscript (text property), 2-50  
 @U, underscore (text property), 2-50  
 @Z, color number, 2-50

## A

A-Page  
   under Master Row decl, 3-7  
   under Master Table decl, 3-3  
 A-Page # Prefix, under Page Number Stream decl, 2-3  
 A-Page Numbering, under Page Number Stream decl, 2-3  
 A-Page Properties, under Component cmd, 2-44  
 A-Page Style, under Page Number Stream decl, 2-3  
 Alignment, under Class decl, 2-21

Allow Page Break After  
   under Class decl, 2-21  
   under Master Row decl, 3-7  
   under Master Table decl, 3-3  
 Allow Page Break After Hyphen, under Page decl, 2-7  
 Allow Page Break Before  
   under Class decl, 2-21  
   under Master Row decl, 3-7  
   under Master Table decl, 3-3  
 Allow Page Break Within  
   under Class decl, 2-21  
   under Master Row decl, 3-7  
   under Master Table decl, 3-3  
 ASCII Unit, under Document decl, 2-4  
 Auto # Stream, under Page Number Stream decl, 2-3  
 Auto Edit  
   under Cell cmd, 3-9  
   under Master Frame decl, 2-31

## B

Balance Columns, under Page decl, 2-7  
 Baseline to Baseline, under Page decl, 2-7  
 Begin New Column  
   under Class decl, 2-21  
   under Master Row decl, 3-7  
   under Master Table decl, 3-3  
 Begin New Page  
   under Class decl, 2-21  
   under Master Row decl, 3-7  
   under Master Table, 3-3  
 Begin on Anchor Page, under Master Frame decl, 2-31  
 Bleed, under Page decl, 2-7  
 Border, under Master Row decl, 3-7  
 Bottom Border Visible, under Master Table decl, 3-3

Bottom Border Weight, under Master Table decl, 3-3

**Bottom Margin**

- under Class decl, 2-21
- under Master Row decl, 3-7
- under Master Table decl, 3-3
- under <!OPS, ...> Declaration, 2-2
- under Page decl, 2-7

**C**

Color, under Cell cmd, 3-9

Column N Left Ruling Visible, under Master Table decl, 3-3

Column N Left Ruling Weight, under Master Table decl, 3-3

Column N Top Ruling Visible, under Master Table decl, 3-3

Column N Top Ruling Weight, under Master Table decl, 3-3

Column N Width (fixed), under Master Table, 3-3

Column N Width (proportional), under Master Table decl, 3-3

Columns, under Page decl, 2-7

Component Bar Width, under Document decl, 2-4

Composition, under Class decl, 2-21

Consecutive Hyphens, under Page decl, 2-7

Contents, under Class decl, 2-21

Count, under Index cmd, 2-56

**D**

Decimal Char, under Class decl, 2-21

Decimal Tab, under Class decl, 2-21

Default Page Stream Name, under Document decl, 2-4

Default Printer, under Document decl, 2-4

Depth at Page Break, under Page decl, 2-7

Depth No Page Break, under Page decl, 2-7

Diagram, under Master Frame decl, 2-31

Dictionary, under Index cmd, 2-56

Doc (Index Document Name), under Index cmd, 2-56

Double Sided, under Document decl, v, 2-4

**E**

End on Anchor Page, under Master Frame decl, 2-31

**F**

Facing Pages, under Document decl, 2-4

Feathering, under Page decl, 2-7

**Fill**

- under Class Defaults decl, 2-19
- under component cmd, 2-42

Final Output Device, under Document decl, 2-4

First Indent, under Class decl, 2-21

First Page, under Page decl, 2-7

Float Precision, under Document decl, 2-4

**Font**

- under Class decl, 2-21
- under component cmd, 2-42
- under Master Row decl, 3-7

Font Unit, under Document decl, 2-4

Footer, under Master Row decl, 3-7

Footer Border Visible, under Master Table decl, 3-3

Footer Border Weight, under Master Table decl, 3-3

Frame Margin Stretch, under Page decl, 2-7

Frozen Number Streams, under Page decl, 2-7

**G**

Gutter, under Page decl, 2-7

**H**

Header, under Master Row decl, 3-7

Header Border Visible, under Master Table decl, 3-3

Header Border Weight, under Master Table decl, 3-3  
 Header Page, under <!Document, ...> Declaration, v, 2-4

Height  
     under Master Frame decl, 2-31  
     under Page decl, 2-7

Hidden  
     under component cmd, 2-42  
     under Row cmd, 3-8

Horizontal Alignment, under Master Frame decl, 2-31

Horizontal Reference, under Master Frame decl, 2-31

Hyphenation  
     under Class decl, 2-21  
     under Page decl, 2-7

## I

Indent Count, under Class decl, 2-21

Inner Margin, under Page decl, 2-7

## K

Kerning properties, under component cmd, 2-51

## L

Left Border Visible, under Master Table decl, 3-3

Left Border Weight, under Master Table decl, 3-3

Left Margin  
     under Class decl, 2-21  
     under Master Table decl, 3-3  
     under Page decl, 2-7

Left Profile, under Class decl, 2-21

Left Ruling Color, under Cell cmd, 3-9

Left Ruling Visible, under Cell cmd, 3-9

Left Ruling Weight, under Cell cmd, 3-9

Left Tab, under Class decl, 2-21

Letterspace Max, under Class decl, 2-21

Letterspacing, under Class decl, 2-21

Line Spacing, under Class decl, 2-21

Line Spacing Unit, under Document decl, 2-4

## M

Manual Sheet Feed, under Document decl, v, 2-4

Margin Shrink, under Page decl, 2-7

Margin Stretch, under Page decl, 2-7

Margins, under Page decl, 2-7

Max Feathering, under Page Number Stream decl, 2-7

Maximum Page #, under Page Number Stream decl, 2-3

Measurement Precision, under Document decl, 2-4

Measurement Unit, under Document decl, 2-4

## N

Name  
     under Master Frame decl, 2-31  
     under Page Number Stream decl, 2-3

No Border, under Master Frame decl, 2-31

Not Selectable  
     under Cell cmd, 3-9  
     under Master Frame decl, 2-31

Numbered, under Master Frame decl, 2-31

Numeric Tab, under Class decl, 2-21

## O

On Anchor Page, under Master Frame decl, 2-31

Orientation Inverted, under Document decl, 2-4

Orphan Control  
     under Class decl, 2-21  
     under Master Table decl, 3-3

Outer Margin, under Page decl, 2-7

Overlap, under Master Frame decl, 2-31



## P

Page # Prefix, under Page Number Stream decl, 2-3  
Page # Prefix Two, under Page Number Stream decl, 2-3  
Page # Stream Name, under Index cmd, 2-56  
Page # Style, under Page Number Stream decl, 2-3  
Pattern, under Cell cmd, 3-9  
Placement, under Master Frame decl, 2-31  
Points Precision, under Document decl, 2-4  
Print Deletion Marks, under Document decl, 2-4  
Print Rev Bars, under Document decl, v, 2-4  
Print Strikes, under Document decl, 2-4  
Print Underlines, under Document decl, 2-4  
Profiling, under Class decl, 2-21

## R

Read Only  
    under Cell cmd, 3-9  
    under component cmd, 2-42  
    under Master Row decl, 3-7  
    under Row Cmd, 3-8  
Repeating, under Master Frame decl, 2-31  
Revision Bar Placement, under Page decl, 2-7  
Right Border Visible, under Master Table decl, 3-3  
Right Border Weight, under Master Table decl, 3-3  
Right Margin  
    under Class decl, 2-21  
    under Page decl, 2-7  
Right Tab, under Class decl, 2-21  
Row, under Master Table decl, 3-3  
Rulings to Bottom, under Master Table decl, 3-3

## S

Same Page, under Master Frame decl, 2-31  
See, See Also, under Index cmd, 2-56

Shared Contents, under Master Frame decl, 2-31  
Size Contents to Height, under Master Frame decl, 2-31  
Size Contents to Width  
    under Cell cmd, 3-9  
    under Master Frame decl, 2-31  
Sort, Sort String, under Index cmd, 2-56  
Spot Color Separation, under Document decl, 2-4  
Starting Page #, under Page Number Stream decl, 2-3  
Straddle  
    under Class decl, 2-21  
    under Master Table decl, 3-3  
Straddle (vertical), under Cell cmd, 3-9  
Straddle N, under Cell cmd, 3-9  
Style = Metric, under <!OPS, ... > Declaration, 2-2  
Style Fun, Style Name, Page# Sep  
    Bottom Margin, under <!OPS, ...> Declaration, 2-2  
    Inner Margin, under <!OPS, ...> Declaration, 2-2  
    Left Margin, under <!OPS, ...> Declaration, 2-2  
    Outer Margin, under <!OPS, ...> Declaration, 2-2  
    Page Height, under <!OPS, ...>, 2-2  
    Page Width, under <!OPS, ...> Declaration, 2-2  
    Right Margin, under <!OPS, ...> Declaration, 2-2  
    Top Margin, under <!OPS, ...>, 2-2  
Subcomponent, under component decl, 2-45  
Superscript, under Master Frame decl, 2-31

## T

Tab Origin, under Class decl, 2-21  
Table Page Break Rulings, under Master Table decl, 3-3  
To Named, under Index cmd, 2-56  
To Next, under Index cmd, 2-56  
TOC Doc Name, under Class decl, 2-21  
TOC Page Stream, under Class decl, 2-21  
Top Border Visible, under Master Table decl, 3-3  
Top Border Weight, under Master Table decl, 3-3

Top Margin  
     under Class decl, 2-21  
     under Master Row decl, 3-7  
     under Master Table Declaration, 3-3  
     under Page decl, 2-7

Top Ruling Color, under Cell cmd, 3-9

Top Ruling Visible, under Cell cmd, 3-9

Top Ruling Weight, under Cell cmd, 3-9

Track Kern Spaces, under Class decl, 2-21

Turned Pages, under Page decl, 2-7

Typeface, under Index cmd, 2-56

## U

Underline at Descender, under Document decl, 2-4

## V

Vert Just, under Page decl, 2-7

Vertical Alignment  
     under Cell cmd, 3-9  
     under Master Frame decl, 2-31

Vertical Margins, under Page decl, 2-7

Vertical Reference, under Master Frame, 2-31

## W

Widow Control  
     under Class decl, 2-21  
     under Master Table decl, 3-3

Width  
     under Master Frame, 2-31  
     under Page decl, 2-7

Wordspace Max, under Class decl, 2-21

Wordspace Min, under Class declaration, 2-21

Wordspace Nom, under Class decl, 2-21

Wordspacing, under Class decl, 2-21

## Z

Zoom, under Document decl, 2-4

# Special Cases

## A

A-Page Numbering, declared under Page Number Stream, 2-3

A-page prefixes, declared under Page Number Stream declaration, 2-6

Angle bracket, as text character in Interleaf ASCII markup, 1-7

## C

Command names in markup, use of quotation marks, 1-9

Comments, not allowed in diagramming markup, 1-7

Component names, cannot be duplicated, 2-20

Components vs Frames, differences in Interleaf ASCII markup, 2-32

Coordinates in Graphics Objects markup, number of digits permissible, 4-8

## E

End Table cmd, 3-11

End Text, required in microdocument markup, 4-20

## F

Fill Property, special use of, 2-19

Fill Property for components, special use of, 2-43

<Fn> font tokens, do not accept text property inheritance, 2-47

<FO> font token, does not accept text property markup, 2-52

Font inheritance

as a special case, 2-46  
special constraints, 2-49

Font inheritance procedure, 2-29

Font number assignment procedure, D-4

Font Token, importance in inline component markup, 2-46

Frame width constraints, 2-36

Frames inside table cells, special procedures for, 3-10

## K

Kerning, as text property of the font, 2-52

## N

No Dictionary property, facilitates loading of large files, 2-51

## P

Page vs Microdocument Properties, 2-7

## Q

Quotation Marks, inside quotation marks, 1-9

## T

Tabs, cannot insert if alignment Centered or Right, 2-23