

# CHAPTER 2

## **FOSI Basics**

---

The fundamentals of FOSI are covered in this chapter, including a description of the formatting process. Technical details about the language are provided in subsequent chapters.

- **FOSI standard**, page 6
- **FOSI interfaces**, page 6
- **Screen and print FOSIs**, page 6
- **FOSI compilation**, page 6
- **FOSI structure**, page 7
- **FOSI formatting properties**, page 8
- **Input and output structures**, page 25
- **Formatting process** page 27
- **Specifying FOSI characteristics and defaults**, page 28
- **Resolving FOSI formatting specifications**, page 38

**JARGON** 

FOSI stands for "Formatting Output Specification Instance," where "Instance" refers to the fact that a FOSI is technically an SGML document that contains elements and attributes — but no #PCDATA — in conformance with the Output Specification DTD.

**TIP** 

Anecdotal evidence suggests that a separate screen FOSI may slightly reduce the time needed to load a long document into Arbortext Editor.

## FOSI standard

FOSI was developed by the U.S. Department of Defense to format SGML and XML technical data. The FOSI standard is described in *MIL-PRF 28001C Appendix B*.

FOSI specifies formatting for structured markup documents, including XML, SGML and HTML. FOSI stylesheets are written in SGML.

**NOTE:** Arbortext Editor does not support everything in the FOSI standard, which is documented in the annotated standard in *Arbortext-path\tutorials\28001C\28001C.sgm*. Arbortext Editor's version of the OutSpec DTD is reproduced in this book, beginning on page 765.

## FOSI interfaces

Arbortext Editor provides two interfaces for FOSI development: the tagged FOSI interface and the FOSI style panels interface. They are best used in tandem, as each has major strengths and a few limitations, which may be referred to in this book. However, this book does not describe or teach the use of the FOSI interfaces.

## Screen and print FOSIs

It may be desirable to have different formatting for the Edit window display and print/PDF output. Arbortext Editor supports this. For example, a font designed for screen display can be coded for the Edit window without affecting the font used for print/PDF output. The formatting for each can be coded in the same FOSI or in separate FOSIs. A FOSI designed just for Edit window display is referred to as a "screen FOSI," while a FOSI used just for print/PDF output is called a "print FOSI."

## FOSI compilation

FOSI is a compiled language. During FOSI development, the FOSI must be compiled for new coding to take effect. This usually takes only a second or two. New formatting that applies to the Edit window is displayed immediately.

Formatting that applies only to print/PDF output is seen when the document is previewed or printed.

**NOTE:** When an SGML/XML document is opened in Arbortext Editor, its FOSI is automatically compiled. When a different FOSI is selected, it is also automatically compiled.

If FOSI compilation fails, a default FOSI that contains minimal formatting, *Arbortext-path\lib\dfmt.fos*, is used instead.

**NOTE:** The ACL command `set fosiwarnings=on|off` controls whether FOSI compilation warnings are displayed. This can be set in Preferences.

## FOSI structure

A FOSI is an SGML document that conforms to the OutSpec DTD, an annotated version of which is located on page 765. A valid FOSI contains no #PCDATA, only elements and attributes. Most elements are singletons, but some are tag pairs, including container elements with no attributes.

The default FOSI for a document type has the same filename as the DTD plus the `.fos` filename extension for FOSI. Thus, `docbook.fos` is the default FOSI for `docbook.dtd`.

Arbortext adds a processing instruction similar to the following at the beginning of a `.fos` file. However, a FOSI is valid without it.

```
<!--Arbortext, Inc., 1988-2007, v.4002-->
```

Next in a `.fos` file is the public identifier for the OutSpec DTD, which is required for Arbortext Editor to recognize the file as a FOSI. It is followed by any internal entity declarations. For example:

```
<!DOCTYPE OUTSPEC PUBLIC
"-//ArborText//DTD OUTPUT SPEC MIL-PRF-28001 REV B AMEND 1 19961231//EN" [
<!ENTITY train SYSTEM "train.tif" NDATA tif>
<!NOTATION tif SYSTEM "tif">
]>
```

The next line is an optional processing instruction that identifies the FOSI.

```
<?APT StylesheetID Title="Manual" Description="Owners Manual"
CompositionTypes="print,pdf,htmlfile">
```

### FOSI TIP

The number of some FOSI components, such as page layouts, that are supported by FOSI compilation is limited. Please contact Arbortext Technical Support for specifics.

### TIP

You can use the DTD Viewer to see the OutSpec DTD structure. Enter `dtviewer` at the command line in the tagged FOSI editor.

### TIP

FOSI is written in SGML, which looks very similar to XML. The major difference is how singleton elements are represented.

After the top tag in the `outspec.dtd`, which is `<outspec>`, Arbortext Editor adds a processing instruction that indicates the FOSI is valid for product versions from Adept 8.0 forward. This PI is optional; a FOSI is valid without it.

```
<outspec><?Pub Lcl _label="Adept v8.0">
```

The rest of a FOSI stylesheet is organized into various **descriptions**. Each description provides a way to specify relevant formatting properties.

**NOTE:** As a declarative language, FOSI defines, or describes, the problem to be solved and leaves the solution to the interpreting software. FOSI describes the desired formatting — the “what” — while leaving the “how” to the formatter. In other words, FOSI does not provide a way to tell the formatting system “first do this, then do that.”

FOSI descriptions are listed below with the page number where each is discussed in detail in this book. Note that Arbortext Editor provides proprietary solutions for graphics and tables, and does not support FOSI `graphdesc` or `tabdesc`. Consequently, they are not covered in *Practical FOSI*.

- `rsrctdesc` (resource description), page 192
- `secdesc` (security description), page 224
- `pagedesc` (page description), page 231
- `styldesc` (style description), page 302
- `docdesc` (document description), page 305
- `envdesc` (environment description), page 306
- `ftndesc` (footnote description), page 506
- `tabdesc` (table description)
- `grphdesc` (graphic description)

## FOSI formatting properties

The FOSI standard's term for what is generically called a formatting property is **characteristic**. A characteristic is an attribute in the OutSpec DTD. Characteristics are arranged in groups called **categories**. A category is an element — tag pair or singleton — in the OutSpec DTD.

Examples of FOSI categories are `font`, `ruling`, `enumerat`, and `gutter`. The `font` category, like most categories, contains several characteristics, including a `size` characteristic to specify the type size. The `gutter` category, by contrast,

has a single characteristic called `width`, which specifies the extent of the gutter between columns of flowing text on a page.

Some categories are termed **nonrepeating categories** because they can be specified just once for an element. For instance, `font` and `leading` are nonrepeating categories because more than one of them doesn't make sense — text can't be displayed in two different fonts at the same time.

Categories such as `ruling`, on the other hand, may be specified as many times as needed. For example, two horizontal rules can be output before the content of a `<title>` element and two horizontal rules can be output after the same title. Such categories are called **repeating categories**.

Some nonrepeating categories are **inheriting categories**, which means they have an `inherit` attribute in the OutSpec DTD. During the formatting process, they may inherit formatting properties from ancestor elements. Categories without an `inherit` attribute are termed **defaulting categories**. The mechanics of inheritance and defaulting are detailed beginning on page 41.

The FOSI standard defines two kinds of characteristics:

- **composition characteristics**, which apply to elements in the document; for example, the `font` category's characteristics
- **pagination characteristics**, which apply to page layouts defined in the FOSI stylesheet; for example, the `gutter` category's `width` characteristic —

## Composition characteristics

This section introduces composition characteristics used to format document elements. Detailed information about composition categories and characteristics is provided in **Styldesc** (page 302).

**Table 1** lists composition categories and characteristics supported by Arbortext Editor. For nonrepeating categories, the category name is shown in bold. Repeating categories are shown in bold italic. The OutSpec category name is followed by the name used in the FOSI style panels in parentheses, if it is different. The page number where the category is discussed in detail in *Practical FOSI* is shown in square brackets after that. For inheriting categories, `inherit` is listed before other characteristics.

### TIP

The FOSI standard does not provide ways for controlling everything related to formatted output. Some things are left to the formatting engine. For example, there is no way to specify the hyphen character the formatting system inserts when it hyphenates a word.

### JARGON

For the most part, the FOSI standard uses the term `characteristic` for all attributes in the OutSpec DTD. However, some are more properly called attributes rather than characteristics because their values are not formatting properties. Instead, their values affect the mechanics of FOSI formatting. For example, `inherit`, `charsubsetref`, and `envname` are really attributes not characteristics. This book follows the OutSpec's lead and usually uses the term characteristic to apply to everything.

**NOTE:** Some composition characteristics apply only to elements that begin a new line of text and/or end a line of text, as detailed in **Block versus inline elements** on page 15.

**Table 1 Composition categories and characteristics**

<b>algroup</b> (side-by-side) [p. 361] direction hgutter minwdnext postspace refpoint valignnext	<b>boxing</b> [p. 405] boffset brel bthick btype inclr inpct leftgap loffset lthick ltype outclr outpct rightgap roffset rthick rtype siderel thick toffset trel tthick ttype	<b>chgmark</b> (change mark) [p. 411] baroffset barthick cmclass join type	<b>enumerat</b> (enumerate) [p. 450] enumid increm setvalue	<b>float</b> [p. 420] flidref inline pagetype specwidth scope width
<b>font</b> [p. 324] inherit famname offset posture size smallcap style weight width	<b>hight</b> (highlighting) [p. 337] inherit allcap bckclr fontclr forpct scorechr scorechron scoreoff scorespc scorewt scoring xscoreoff xscorewt xscoring	<b>hyphen</b> (hyphenation) [p. 333] inherit lang hyph zone	<b>indent</b> [p. 349] inherit firstln leftind rightind	<b>keeps</b> [p. 397] floatsout keep next orphanct prev scope widowct

*continued . . .*

<b>leading</b> [p. 329] inherit lead	<b>link</b> [p. 433] href	<b>postsp</b> (postspace) [p. 386] condit maximum minimum nominal priority	<b>presp</b> (prespace) [p. 382] condit maximum minimum nominal priority	<b>putgraph</b> (put graphic) [p. 445] depth graphname hoffset hscale placement rotation scalefit voffset vscale width
<b>quadding</b> (justification) [p. 345] inherit lastquad quad	<b>reset</b> [p. 455] resetlist	<b>ruling</b> [p. 436] lentype placement rellen ruleclr speclen thick type	<b>savetext</b> (save text) [p. 460] append conrule placemnt textid xrefidtag	<b>sentxsp</b> (sentence spacing) [p. 374] inherit maximum minimum nominal
<b>span</b> (columns) [p. 357] pageflow span	<b>suppress</b> [p. 341] sup	<b>textbrk</b> (text break) [p. 389] endln htmltag newpgmdl noalignprev pageid startcol startln startpg	<b>usetext</b> (use text) [p. 474] placemnt source useparam userule	<b>wordsp</b> (word spacing) [p. 378] inherit avoidoverfull maximum minimum nominal

## Element formatting

In a FOSI stylesheet, composition categories and characteristics are specified in an **element-in-context** (e-i-c). An element matches at most one e-i-c in a FOSI. That e-i-c is termed the **best match**. Matching occurs during the formatting process. The element being matched is the **current element**.

## Element-in-context

An e-i-c has three attributes that are used in the process of finding the best match for the current element. They are:

- the **generic identifier**, which contains one or more element names. For example:

```
<e-i-c gi="warning">
<e-i-c gi="filename pathname">
```
- The **context** of the element in the document structure, which is a string that may include one or more wildcards. For example:

```
<e-i-c gi="item" context="list">
```
- The position of the element within its parent element, termed **occurrence**, whose value may be one of the following:
  - ▶ all (the default)
  - ▶ first
  - ▶ notfirst
  - ▶ last
  - ▶ notlast
  - ▶ middle
  - ▶ only

For example:

```
<e-i-c gi="item" context="inline-list" occur="first">
<e-i-c gi="item" context="inline-list" occur="middle">
<e-i-c gi="item" context="inline-list" occur="last">
```

## Context and occurrence

The same generic identifier can be specified in different `e-i-c`s with different context and/or occurrence in order to specify different formatting accordingly. For example, the formatting for a `<title>` element may differ according its context. A `<chapter>` `<title>` may be centered while a `<section>` `<title>` is flush left. To support this, more than one `e-i-c` is needed in the FOSI, each with a different context string.

Similarly, an element's occurrence within its parent element can specify different formatting. For example, the first `<paragraph>` after a `<chapter>` `<title>` may be flush left while all other `<paragraph>`s are indented. Another example: when there is only one item in a `<number-list>`, an `e-i-c` with occurrence set to `only` can be coded to output a bullet instead of numbering.

**NOTE:** Occurrence applies to the element's position within its parent element, regardless of what is coded in the `context` attribute of the `e-i-c`. If the

context attribute is not specified, occurrence applies to every context that is matched.

Two of the examples below include a wildcard (\*) in the context string. See **Best match algorithm** below and **Context syntax** on page 312 for further detail.

### Figure 1 Examples of context and occurrence

```
e-i-c gi="title" context="chapter"
e-i-c gi="title" context="figure"
e-i-c gi="title" context="table"

e-i-c gi="para" context="* intro"
e-i-c gi="para" context="* body"

e-i-c gi="para" context="chapter body" occur="first"
e-i-c gi="para" context="chapter body" occur="notfirst"

e-i-c gi="number-item" context="number-list" occur="only"
e-i-c gi="number-item" context="number-list" occur="all"

e-i-c gi="listitem" occur="first"
e-i-c gi="listitem" occur="middle"
e-i-c gi="listitem" occur="last"
e-i-c gi="listitem" occur="only"

e-i-c gi="row" context="tbody tgroup table" occur="notlast"
e-i-c gi="row" context="tbody tgroup table" occur="last"
```

**NOTE:** Arbortext Editor supports specifying an XPath pattern instead of using the context and occur attributes. The XPath expression should select the element or elements specified by the gi attribute. However, if any e-i-c's in a FOSI use XPath, the rsrcdesc attribute eicorderispriority must be set to "1", which means that the e-i-cs for an element are evaluated in the order listed in the FOSI, and the first one that matches is used.

## Element-in-context matching

If a FOSI has just one e-i-c for an element name, and its context and occur attributes are not specified, that e-i-c matches every element of that name in the document. For example, the formatting for a <partnumber> element may be the same in all contexts. In such cases, only one e-i-c is needed in the FOSI. It is the best match that matches every occurrence of <partnumber> in every context in the document.

### FOSI TIP

If occurrence is not sufficient to match an element's position within its parent, a FOSI can be coded to count elements and apply formatting accordingly.

**Programming pseudo-elements** (page 116) and **#FOSI** (page 92) can be used for this purpose.

If no `e-i-c` in the FOSI matches an element in the document, the formatter assumes a matching `e-i-c` with an empty `charlist`. However, an `e-i-c` with the keyword `#MISSING` for the generic identifier can be coded in the FOSI for use when there is no match for the current element.

If the FOSI has more than one `e-i-c` with the same `gi`, `context`, and `occur` attributes, any duplicates `e-i-cs` are reported when the FOSI is compiled. The first `e-i-c` is used; duplicates are ignored.

## Best match algorithm

An algorithm described in the Outspec (B.3.6.2.2.2) defines the rules that determine which `e-i-c` is the best match. It is covered in this section for those who want to know the gory details.

The algorithm compares the context of the current element (the element being processed) with the context strings in `e-i-cs` in the FOSI to find `e-i-cs` that are possible matches.

**NOTE:** At this point, wildcard characters (\*) in `e-i-c` context strings are ignored. However, if no possible matches are found for the current element's context, a second comparison is made that includes wildcards.

In the OutSpec example, the current element to be matched has the following context: `para item randlist para step1 paar0 chapter`

The `e-i-cs` with the same name in the FOSI as the current element in the document have the following context strings:

```
context="para"  
context="para item"  
context="para * item"  
context="para * para0"  
context="para * randlist * chapter"  
context="* para step1 * chapter"
```

The strings are compared in the grid below. For each matching element, 1 (one) is appended to a decimal. For each element with no match and for each wildcard (\*), a 0 (zero) is appended to the decimal. The highest number, in this case .11, indicates the context string that best matches the element's context in the document.

**Table 2 Matching context path to context string**

	para	item	randlist	para	step1	para0	chapter	=
<b>e-i-c context strings</b>	para							.1
	para	item						.11
	para	* item						.101
	para	0	0	0	0	* para0		.1000001
	para	0	* randlist	0	0	0	* chapter	.100100001
	0	0	0	* para	step1	0	* chapter	.000011001
KEY Append to the decimal: 1 for each matching element, 0 for each 0, and 0 for each wildcard (*). The highest number determines the best match.								

## Scope of formatting environment

The formatting characteristics in an e-i-c create a formatting environment that is in effect during the scope of the element that matches it. The start tag of the element in the document begins the scope, and the end tag of the element closes or terminates the scope, which causes the scope of the parent element to be resumed.

During the scope of any element, the start tag of a child element defines a new scope with a new and potentially different formatting environment that remains in effect until the end tag of that child element.

Anything generated by an e-i-c is considered to be within the scope of that e-i-c and is affected by the formatting environment associated with that e-i-c. This applies to empty elements as well as elements with content.

## Block versus inline elements

FOSI makes a distinction between inline and block elements. A **block element** starts a new line and forces a new line after it ends. A block element is usually distinguished by vertical space before and after it. In addition, a block element may be indented on the left and/or right. <paragraph> and <figure> are examples of block elements. An **inline element** is part of the flowing text, with no extra space before or after it. An inline element does not start or end a line. <bold> and <partno> are examples of inline elements.

The `textbrk` category specifies line-breaking behavior. A block element is coded in a FOSI as `textbrk startln="1" endln="1"`. An inline element is coded as `textbrk startln="0" endln="0"`. The other possibilities are: `textbrk startln="1" endln="0"` (start a new line but do not force a new line after the element content) and `textbrk startln="0" endln="1"` (do not start a new line but do force a new line after the element).

The `leading`, `indent`, `quadding`, `presp`, and `postsp` categories are irrelevant for inline elements. For example, when `presp` and `postsp` categories are coded for an inline element such as `<bold>`, they have no effect because `presp` requires `textbrk startln="1"`, and `postsp` requires `textbrk endln="1"`. The following lists show the formatting characteristics that require `startln="1"` and/or `endln="1"`, respectively.

**Categories/characteristics requiring `textbrk startln="1"`**

- `indent firstln`
- `leading`
- `presp`
- `quadding`
- `keeps` (for column and page breaking; does not apply to line breaking)

**Categories/characteristics requiring `textbrk endln="1"`**

- `indent leftind`
- `indent rightind`
- `postsp`
- `quadding` (overrides `quadding` for containing element)

**NOTE:** The values for the `indent`, `quadding`, and `leading` category characteristics are fixed for the duration of a given block structure.

**NOTE:** An `e-i-c` with `algroup` is treated as if it were coded with `textbrk startln="1" endln="1"`.

## Flowing text

As each element is formatted by its matching `e-i-c`, the element's content (if any) and anything generated by the `e-i-c` is added to the flowing text output stream managed by the formatting engine. Flowing text flows from one column to the next on a page and, when that page is filled, to the next page.

Page formatting is discussed on page 19. First, pagination characteristics and their categories are discussed.

## Pagination characteristics

This section introduces pagination characteristics used to format **layout areas** such as **Column Area** and **Footnote Area** ( page 22). Detailed information about pagination categories and characteristics is provided in the section on **Pagedesc** (page 231).

Pagination characteristics define:

- the physical appearance of pages, including size, margins, and areas for placement of text and/or graphics
- where appears in header, footer, page regions, and change mark areas on a page
- what pages without content (blank pages) should look like
- where floated elements may occur on a page
- which numbering should be incremented and/or output
- which numbering should be saved for output in flowing text

**NOTE:** Inheritance, which is discussed on page 41, does not apply to the specification of pagination characteristics. The FOSI must supply values for characteristics of a Layout Area. Also, Layout Areas do not contribute to the ancestry of `e-i-cs`.

Pagination characteristics supported by Arbortext Editor are shown in **Table 3**. The category name is shown in bold, followed by the term used in the FOSI style panels interface in parentheses, if different. The page number where the category is discussed in detail is shown in square brackets. Container elements with no characteristics are included.

**Table 3 Pagination categories and characteristics**

<b>blankpg</b> (blank page) [p. 246] botfloat chgmkplc hgmkoff layoutconfig maxfloatpct nomdepth topfloat xvjstretch width	<b>botmarg</b> (bottom margin) [p. 259] nomdepth	<b>column</b> [p. 291] botfloat topfloat vjprior width	<b>flowtext</b> [p. 284] balance botfloat chgmkplc leftind numcols rightind topfloat	<b>footer</b> [p. 269] nomdepth spaflow
<b>footnote</b> [p. 298] ftnfloat ftnsepln ftnsepth spabove width	<b>gutter</b> [p. 295] width	<b>header</b> [p. 265] nomdepth spaflow	<b>leftmarg</b> (left margin) [p. 261] width	<b>pageref</b> (page reference) [p. 254] pgidref
<b>pageres</b> (page folio resources) [p. 250]	<b>pagespec</b> (page specification) [p. 252] pgid	<b>pageset</b> (page layout set) [p. 232] id blankpg orient	<b>rectobb</b> (recto blank back) [ <b>Rectobb</b> on page 248]	<b>rectopg</b> (recto) [p. 240] botfloat chgmkplc chgmloff layoutconfig maxfloatpct nomdepth topfloat xvjstretch width

*continued . . .*

<b>region</b> [p. 272] width bckclr height oalign layer rotation valign xoff yoff	<b>rtmarg</b> (right margin) [p. 263] width	<b>sectext</b> (security text) [p. 280] scope	<b>topmarg</b> (top margin) [p. 257] nomdepth	<b>versobf</b> (verso blank front) [p. 249]
<b>versopg</b> (verso) [p. 244] botfloat chgmkplc chgmkoff layoutconfig maxfloatpct nomdepth topfloat xvjstretch width				

## Page formatting

Document elements are formatted by composition characteristics and become flowing text that fills columns and pages. Columns and pages are formatted by pagination characteristics.

Each page may use a page layout that is not necessarily the same as the page layouts of previous or following pages. For instance:

- the front matter of a document has a single column while the pages in the body of the document contain two columns of flowing text
- the first page of a chapter is a recto page with a deep top margin and no floating allowed to the top of the page
- the page number is flush right on recto (righthand) pages and flush left on verso (lefthand) pages
- wide tables and graphics are output on landscape rather than portrait pages like the rest of the document
- oversize material is output large foldout pages
- blank pages are totally blank, without the page headers and footers that appear on pages with content

**TRIVIA** 

The page models in a `pageset` should have the same page dimensions. However — although there does not appear to be any practical use for this — it is possible to define a `recto` page with a backing `verso` page that has different page dimensions (for example, an 8½×11 `recto` page with a backing 5×5 `verso` page) as long as the column width is the same for both.

## Pagesets and page models

In a FOSI, one or more **pagesets** with one or more **page models** are used to describe the desired page layouts.

**NOTE:** Page orientation (portrait or landscape) is set for a `pageset`, so all page models in a `pageset` have the same orientation.

Each page model can have its own values for the following formatting properties:

- page width
- page depth
- top margin
- bottom margin
- left margin
- right margin
- header
- footer
- vertical space between header and flowtext
- vertical space between flowtext and footer
- number of columns
- column balancing
- gutter width (horizontal space between columns)
- left indent
- right indent
- placement of floated objects
- footnote placement
- change bar placement
- overlays and underlays
- setting for vertical justification
- settings for prioritizing layout components

## Blank pages

If specified in the FOSI, blank pages are automatically inserted by the formatter when a forced page break creates a blank backing or facing page. Blank page models do not contain document content but may output text and graphics from the header, footer, and overlay/underlay regions. For example, “This page is intentionally blank” could appear on an otherwise blank page.

Some documents require the page before a blank page to identify the following page as blank. For example: “Next page blank.” FOSI provides page models for recto pages with a blank back and for verso pages with a blank front so a different header, footer, and/or page region can be specified for those cases. The formatting engine automatically uses the appropriate page model(s) when blank pages occur.

## Quintuples

FOSI provides the following page models:

- recto page (`rectopg`)
- verso page (`versopg`)
- blank page (`blankpg`)
- recto pages with a blank back (`rectobb`)
- verso pages with a blank front (`versobf`)

The five page models are arranged in a quintuple in the order listed above. One recto page model is required, the others are optional.

More than one quintuple may be specified in a pageset. The first quintuple provides the first page model used. The second quintuple provides the next page model used, and so on until the last quintuple is reached. At that point, the page models in the last quintuple are reused until the end of the document or another pageset is called.

## Calculated column widths and depths

The formatting engine uses the values specified for certain pagination characteristics to calculate the column width and flowtext depth for each page model. The page models in a pageset may have different flowtext depths, but their calculated column widths must be the same. Otherwise, a FOSI Compile Warning Message is displayed that “composition cannot support a change in `<columnwidth>` within a pageset.”

For example, a bound document may require wider inner margins (left margin on recto pages, right margin on verso pages). The margins on recto pages and verso pages must be set so the calculated column widths are equal.

The calculated flowtext depth, on the other hand, can be different for each page model in a pageset. An example is when the first page of a chapter begins with

a block of vertical white space before the content, which results in a `flowtext` depth that is less than the depth of the rest of the pages in the pageset.

## Page model references

FOSI supports references to an existing page model. For instance, a blank page may be defined once and referenced in other pagesets so the same blank page layout is used throughout the document.

## Page numbering

Each page model has an optional mechanism for handling page numbering tasks, including incrementing numbering, resetting numbering, and supporting references to page numbers.

## Page Layout Areas

Each page model contains Page Layout Areas (**Figure 2** on the next page), and Float Areas (**Figure 3** on the page after that). The shaded areas rotate for landscape pages.

Figure 2 Page Layout Areas

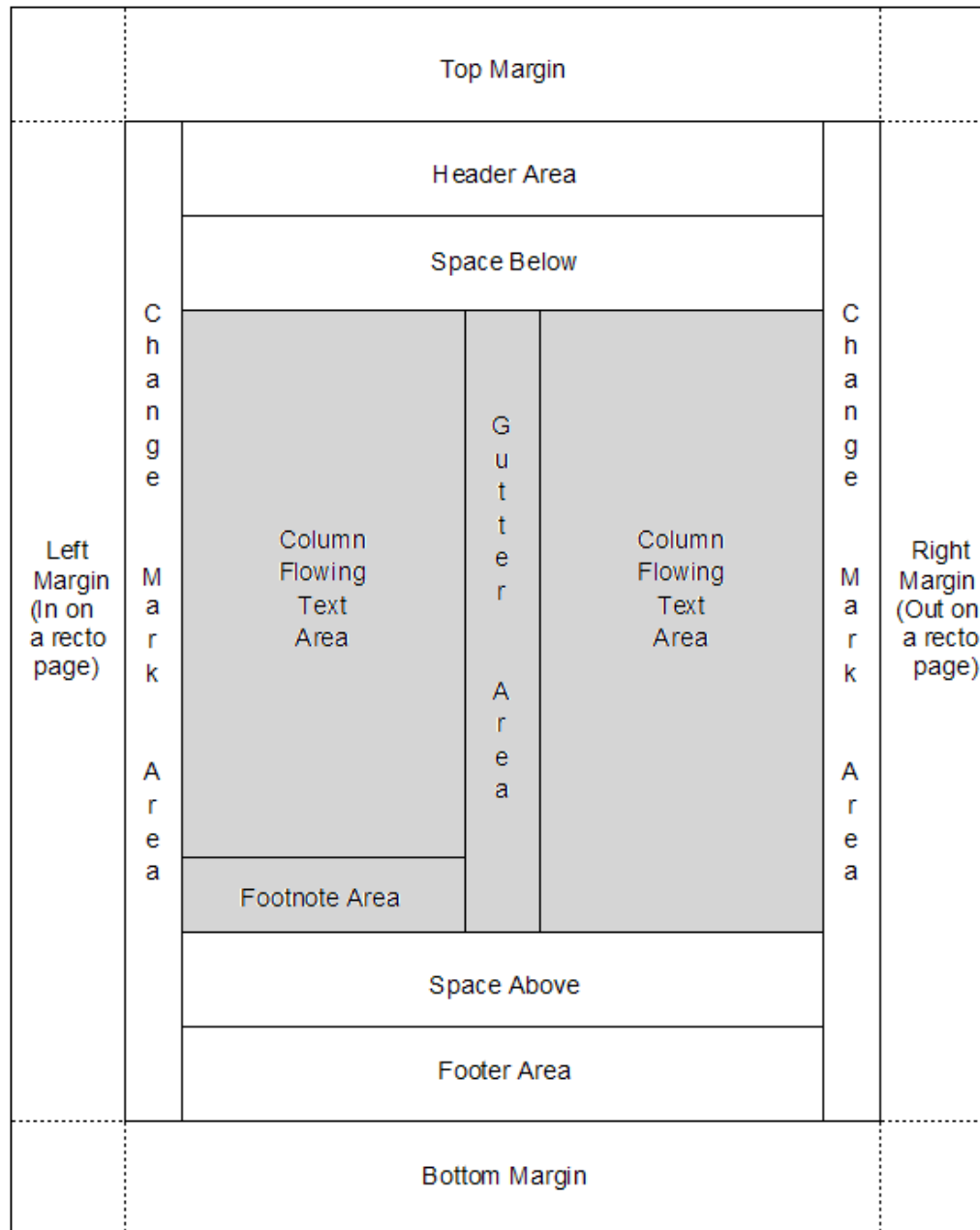
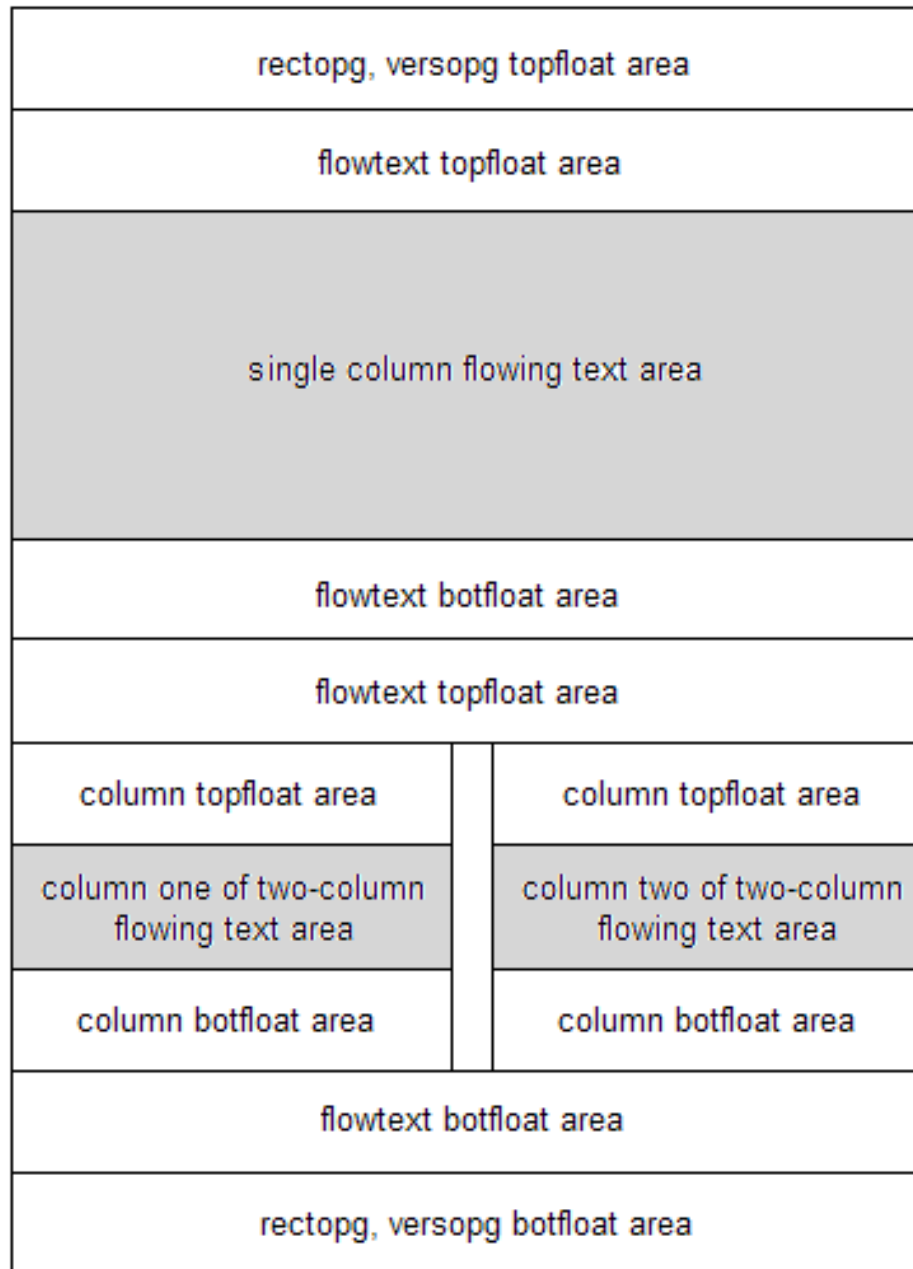


Figure 3 Float Areas



## A note about page fidelity and page integrity

In the OutSpec, **page integrity** is defined as the ability to preserve the same information on each page in a document as it is exchanged between formatting systems. However, this does not necessarily mean that the information will be presented exactly the same way, only that it will appear between the same page boundaries. **Page fidelity** is defined as the ability to preserve the exact presentation characteristics in addition to the same information on pages exchanged between systems.

While the FOSI standard does define a broad set of output characteristics that any compliant formatting engine must support, it leaves much up to the discretion of the formatting engine. For example, the following are left to the formatter to decide, which means that different formatters would produce different results.

- choice of hyphenation dictionaries
- algorithm for optimizing line breaks in a paragraph
- best way to balance columns
- best breakpoint for a page or sequence of pages
- exact placement of floats

Arbortext Editor's formatting engine has evolved over many software releases and produces different results as features are added and, more importantly, as bugs are fixed. Arbortext Editor does not promise page fidelity from one version of the software to the next. However, starting with the 5.3 release, the Arbortext Editor FOSI engine has added extensions to provide for page integrity between releases, which was necessary to support change page applications. However, a description of these extensions is beyond the scope of this book.

## Input and output structures

FOSI does not include an explicit transformation step as do XSL-FO and DSSSL, another formatting language with a transformation process. However, an output tree structure that is different from the structure of the source document can be created with the FOSI categories `suppress`, `savetext`, and `usetext` as well as FOSI **pseudo-elements** and some processing instructions. This useful feature does not require a separate transformation process as with XSL-FO and DSSSL, which is the main reason FOSI is so much faster.

Processing instructions recognized by the formatter are not part of the structure of the input tree. However, any supported PI tag pair does contribute structure to the output tree, and its characteristic settings can be inherited by its descendants in the output tree. (**Figure 289** on **Figure 289** on page 569 provides an example.)

**NOTE:** The formatting engine automatically builds the output structure based on the FOSI coding. This is discussed in **Formatting process** on page 27.

**NOTE:** The input structure determines context for e-i-c matching, but the output structure determines the ancestor/descendant relationships for inheritance in e-i-cs.

## Input and output structures example

Following is a simple example of reordering element content. The content of the first element is suppressed and saved to a string variable for output after the second element. This is also an example of FOSI's ability to generate material for inclusion in the output structure, in this case a comma, a space, and an exclamation point.

### Figure 4 Element order reversed

#### Formatted output

Hello, World!

#### XML fragment

```
<addressee>World</addressee>
<greeting>Hello</greeting>
```

#### FOSI fragment

```
<stringdecl textid="addressee.txt" literal="">
...
<e-i-c gi="addressee">
<charlist inherit="1">
<suppress sup="1">
<savetext textid="addressee.txt" conrule="#CONTENT">
</charlist>
</e-i-c>
<e-i-c gi="greeting">
<charlist inherit="1">
<usetext placemnt="after" souce="\, \,addressee.txt,!\\">
</charlist>
</e-i-c>
```

## Formatting process

This section provides an overview of how the Arbortext Editor formatting engine transforms a source document into an output stream.

When a document is formatted, previewed, or printed, it is traversed by the formatting engine from beginning to end at least once. Beginning with the top tag in the document, the formatter repeats the following process until the top tag's end tag is processed. It:

1. looks for the element-in-context (*e-i-c*) in the FOSI that best matches that element's context in the document and its position within its parent element, as detailed in **Element-in-context matching** on page 13
2. processes the best-match *e-i-c*, as outlined in **Resolving FOSI formatting specifications** on page 38

**NOTE:** Categories with `placemnt="after"` are not processed at this time.

3. moves to the next start or end tag in the input structure and repeats the process

**NOTE:** When an end tag is encountered, any categories in the *e-i-c* with `placemnt="after"` are processed.

If necessary, the formatter makes one or more passes through the document in order to create the desired output stream. For example, many documents include a table of contents (TOC) before the body of the document, which can be generated by a FOSI. However, page number information is not available until after the formatter has made at least one formatting pass through the input document to break it into numbered pages.

During the first formatting pass, the TOC entries and their page numbers for the TOC are accumulated for output during the second formatting pass. However, the insertion of the TOC into the output structure during the second pass may make the entries in the TOC obsolete. For example, consider a document with sequential page numbering on all pages and a FOSI-generated table of contents on the second page of the document. During the first formatting pass, the TOC is not available because the document content after the TOC has not yet been broken into numbered pages.

As the first formatting pass continues, the rest of the document is paged, and each TOC entry with its page number is saved for output in the TOC during the second formatting pass. However, in this case, when the TOC is output, it adds an extra page to the document. This means the sequential page

### TIP

In a production environment, best practice is to perform a completeness check on the structured markup document before formatting. FOSI formatting may not be correct when a document is incomplete and/or out of context. In addition, duplicate IDs and references to missing IDs result in erroneous cross references.

numbering from the first formatting pass no longer matches the entries in the TOC. An additional formatting pass is needed to synchronize page numbers and references to them, including cross references and index entries with page numbers as well as the TOC.

**NOTE:** Arbortext Editor has a formatting pass reduction feature, described on page 528.

## Specifying FOSI characteristics and defaults

An `e-i-c` consists of two parts:

- characteristic list (`charlist`)
- optional, repeatable attribute specification (`att`)

Characteristics can also be specified in the following FOSI components:

- sub-characteristic list (`subchars`)
- characteristic subset (`charsubset`)

Finally, there are three ways that default values can be set for characteristics that are not specified:

- environment (`envdesc`)
- document description (`docdesc`)
- system defaults

These methods of specifying characteristics and defaults are introduced below.

**NOTE:** Composition characteristics can be specified directly in tag pair categories `chgmark`, `ruling`, `header`, `footer`, and `region`. However, each of them is allowed to contain only a few composition categories. For details, refer to the reference material for the individual categories.

### Characteristic list (`charlist`)

Every `e-i-c` has a characteristic list where repeating and nonrepeating categories can be specified. A characteristic list may reference one or more characteristic subsets, which contribute categories to the characteristic list. A characteristic list may also reference an environment, which supplies default values.

A characteristic list has an `inherit` attribute that provides a control for inheritance of characteristic values from ancestor elements, as described in **Inheritance and defaulting** on page 41.

## Characteristic list example

Following is an example of a charlist with an indent category.

**Figure 5 Charlist with indent category**

```
<e-i-c gi="entry" context="* table" occur="all">
<charlist inherit="1">
<indent inherit="0">
</charlist>
</e-i-c>
```

## Sub-characteristic list (subchars)

A sub-characteristic list is a child element of categories that output text and graphics, specifically:

- `footnote` for page footnotes
- `putgraph` for generated graphics
- `sectext` for generated security classification markings
- `usetext` for generated text

A sub-characteristic list may contain repeating and nonrepeating categories, and it may reference one or more characteristic subsets. The formatting specified in a sub-characteristic list applies only to its parent category.

## Sub-characteristic list example

In this example, the sub-characteristic list contains the nonrepeating `font` category.

**Figure 6 Subchars with font category**

```
<usetext source="Hello, World!">
<subchars>
<font inherit="1" famname="Impact">
</subchars>
</usetext>
```

## Characteristic subset (charsubset)

A characteristic subset is a formatting macro. Characteristic subsets provide a way to parameterize a FOSI so any shared coding is coded once and referenced as needed. One or more characteristic subsets can be referenced by:

- charlist
- subchars
- other characteristic subsets

A characteristic subset does not contain a charlist, but it may contain any categories and characteristics that are allowed in a charlist.

### Characteristic subset examples

The following characteristic subsets — with appropriate values — are useful in most FOSIs. They are used through this book.

#### Figure 7 Useful charsubsets

```
<charsubset charsubsetid="allcaps">
<!-- wordsp is optional; values may differ -->
<wordsp minimum="0.45em" nominal="0.45em" maximum="0.45em">
<hight inherit="1" allcap="1">
</charsubset>
<charsubset charsubsetid="block">
<textbrk startln="1" endln="1">
</charsubset>
<charsubset charsubsetid="gray">
<hight inherit="1" fontclr="#EEEEEE">
</charsubset>
<charsubset charsubsetid="bold">
<font inherit="1" weight="bold">
</charsubset>
<charsubset charsubsetid="caption"
charsubsetref="block center keep-together keep-previous bold">
</charsubset>
<charsubset charsubsetid="center">
<quadding inherit="1" quad="center" lastquad="lcenter">
</charsubset>
<charsubset charsubsetid="endline">
<textbrk startln="0" endln="1">
</charsubset>
<charsubset charsubsetid="hyphen-off">
<hyphen inherit="0" hyph="1">
</charsubset>
<charsubset charsubsetid="hyphen-on">
```

```

<hyphen inherit="1" hyph="1">
</charsubset>
<charsubset charsubsetid="inline">
<textbrk startln="0" endln="0">
</charsubset>
<charsubset charsubsetid="italic">
<font inherit="1" posture="italic">
</charsubset>
<charsubset charsubsetid="keep-next">
<keeps next="7">
</charsubset>
<charsubset charsubsetid="keep-previous">
<keeps prev="7">
</charsubset>
<charsubset charsubsetid="keep-together">
<keeps keep="7">
</charsubset>
<charsubset charsubsetid="left">
<quadding inherit="1" quad="left" lastquad="lleft">
</charsubset>
<charsubset charsubsetid="medium">
<font inherit="1" weight="medium">
</charsubset>
<charsubset charsubsetid="next-page">
<textbrk startpg="next">
</charsubset>
<charsubset charsubsetid="postspace">
<!-- values may differ -->
<postsp minimum="0.5em" nominal="0.5em" maximum="0.5em"
priority="high">
</charsubset>
<charsubset charsubsetid="prespace">
<!-- values may differ -->
<presp minimum="1em" nominal="1em" maximum="1em" priority="med">
</charsubset>
<charsubset charsubsetid="right">
<quadding inherit="1" quad="right" lastquad="lright">
</charsubset>
<charsubset charsubsetid="smallcaps">
<font inherit="1" smallcap="1">
</charsubset>
<charsubset charsubsetid="startline">
<textbrk startln="1" endln="0">
</charsubset>
<charsubset charsubsetid="SUPPRESS">
<!-- optional gray fontclr appears only in the Edit window -->
<hlight inherit="1" fontclr="gray">
<suppress sup="1">
</charsubset>
<charsubset charsubsetid="title">
<!-- this is an example; charsubsetrefs may differ -->

```

```

charsubsetref="block center prespace postspace
keep-together keep-next title-font">
</charsubset>
<charsubset charsubsetid="title-font">
<!-- famname value may differ -->
<font inherit="1" famname="Franklin Gothic Heavy">
</charsubset>
<charsubset charsubsetid="underline">
<highlt inherit="1" scoring="1" ...>
</charsubset>

```

Well-named charsubsets act as comments to an e-i-c, as this example shows.

### Figure 8 Charsubset names act as comments

```

<e-i-c gi="emphasis">
<charlist inherit="1" charsubsetref="inline bold"></charlist>
</e-i-c>
<e-i-c gi="quotation">
<charlist inherit="1" charsubsetref="block center prespace-medium
postspace-small keep-together-column">
...

```

The following example demonstrates charsubsetref and subchars.

### Figure 9 Charsubsetref and subchars

```

<charsubset charsubsetid="AM-PM">
<putgraph placemnt="before" graphname="sun" >
<subchars charsubsetref="startline"></subchars>
</putgraph>
<usetext placemnt="before" source="\Good Morning!\">
<subchars charsubsetref="bold endline">
<font inherit="1" famname="Cooper Black">
</subchars>
</usetext>
<putgraph placemnt="after" graphname="moon" >
<subchars charsubsetref="startline">
<textbrk startln="1">
</subchars>
</putgraph>
<usetext placemnt="after" source="\Good Night!\">
<subchars charsubsetref="endline">
<font inherit="1" famname="Perpetua">
</subchars>
</usetext>
</charsubset>
...
<e-i-c gi="daily-schedule">
<charlist charsubsetref="AM-PM">...</charlist>
</e-i-c>

```

## Attribute specification (`att`)

While `charlist` is required in an `e-i-c`, `att` is optional and repeatable. An attribute specification is used to test the value of an attribute or use the attribute value as a FOSI characteristic. Attributes on the current element or on an ancestor of the current element can be accessed directly with an `attspec`. Attributes on the current element's child or sibling elements, however, cannot be accessed from the current element via an `attspec`.

Categories and characteristics related to attribute specifications supported by Arbortext Editor are shown in **Table 4**. The category name is shown in bold, followed by the name used in the FOSI style panels in parentheses, if it is different. The page number where the category is discussed in detail in *Practical FOSI* is shown in square brackets after that.

**Table 4** Attribute specification categories and characteristics

<b>att</b> (attribute specification) [p. 492] logic	<b>fillval</b> (use value) [p. 494] attloc attname fillcat fillchar	<b>specval</b> (test value) [p. 502] attloc attname attval
--	---	--

**NOTE:** FOSI keywords can be used in attribute rules, as detailed in **Chapter 5**.

When an attribute test with `specval` is successful, the designated nonrepeating and repeating categories are added to the `e-i-c`'s formatting environment. For example:

- When the *booksize* attribute on the top tag is set to `glovebox`, a different `pageset` is used.
- When the *language* attribute on the top tag is set to `French`, French hyphenation is used.

Multiple attributes can be tested with `specval` using `and` or `or`. For example:

- When the *change* attribute on the current tag is set to `add` and the *process-changes* attribute on the top tag is set to `yes`, the element content is underlined.
- When the value of the *security* attribute on the current tag is `classified`, or the value of the *security* attribute on the current tag is `secret`, or the

### JARGON

**Attribute specification** is abbreviated as **attspec** and is also referred to as an **attribute rule**, the terminology is used in the FOSI style panels interface.

**FOSI TIP** 

Attribute rules that occur more than once in a FOSI can be parameterized by putting them in FOSI text entities and referencing them as needed, as illustrated in **Figure 157 Sectext for document and page** on page 281.

value of the *security* attribute on the current tag is `topsecret`, then the content is suppressed from the output structure.

- When the *style* attribute on a `<list>` tag is set to `inline`, a new line is not started at the beginning or end of the `<list>`.

`Fillval` can use the value of an attribute as a FOSI characteristic value, including adding the attribute value to the output stream as content. Here are some examples:

- The value of an attribute named *color* is used for the font color.
- The content of a *date* attribute is output as part of the document.
- The value of a *label* attribute on a list item is used instead of numbering generated by the FOSI.

`Specval` and `fillval` can be combined. For example:

- If the *volume* attribute on the top tag in the document is set to a number greater than 1, the value of the *initialize-pageno* attribute on the top tag is used to initialize page numbering in the document.
- If the *style* attribute on a `<list>` tag is set to `bullet` or to `number`, a hanging indent is applied to the element.

Using keywords, attribute rules can be coded to apply only to the Edit window display, print/PDF output, or HTML output. For example, a screen-friendly font can be used just for the Edit window display while a different font is used for print/PDF output.

**NOTE:** The screen FOSI and the print FOSI can be coded in one `.fos` file, if desired. See **FOSI design** on page 687 for information on deciding whether one or two FOSIs is appropriate for your application.

## Attribute specification examples

In the following examples, notice how a characteristic subset is used with `attspecs`.

In this example, when the `<change>` element's *type* attribute is set to `delete`, the element's content is suppressed.

**NOTE:** The `SUPPRESS`, `allcaps`, and `title` characteristic subsets are defined in **Figure 7 Useful charsubsets** on page 30.

## Figure 10 Attribute value suppresses element content

### XML fragment

```
<change type="delete">
...

```

### FOSI fragment

```
<gi="change">
<charlist>...</charlist>
<att>
<specval attloc="change" attname="type" attval="delete">
<charsubset charsubsetref="SUPPRESS"></charsubset>
</att>
</e-i-c>

```

The next example shows how an attribute value is used as a `usetext` source in order to output the value of the `security` attribute on the `<chapter>` tag.

## Figure 11 Usetext outputs attribute value

### XML fragment

```
<chapter security="secret">

```

### FOSI fragment

```
<gi="chapter">
<charlist>...</charlist>
<att>
<fillval attloc="chapter" attname="security" fillcat="usetext"
fillchar="source">
<charsubset>
<usetext>
<subchars charsubsetref="title allcaps"></subchars>
...

```

Following is another example of using an attribute value as a FOSI characteristic. In this case, the `color` attribute on a `<title>` element is used for the font color.

**NOTE:** Valid color names are listed on page 71.

## Figure 12 Attribute value used as `highlight fontcolor`

### XML fragment

```
<title color="blue">Introduction</title>

```

### FOSI fragment

```
<gi="title">

```

```

<charlist charsubsetref="title">...</charlist>
<att>
<fillval attloc="title" attname="color" fillcat="highlt"
fillchar="fontclr">
<charsubset>
<highlt inherit="1">
</charsubset>
</att>
</e-i-c>

```

## Environment (envdesc)

An environment provides a way to specify a default value to be used when no value is available for an empty characteristic. If the current *e-i-c* references an environment, that is the first place the formatter looks for default values. An environment contains a characteristic list, but sub-characteristic list is not allowed.

**NOTE:** An environment can be referenced by one or more *e-i-cs*. However, only one environment can be referenced by an *e-i-c*, and an environment cannot reference another environment.

### Environment examples

In this example, a red 6-pica long, 5-point thick rule is output before and after a `<list>`. The environment provides default values for the missing characteristics in the ruling categories in the *e-i-c* for `<list>`.

**NOTE:** The `textbrk` categories must be coded in the *e-i-c*. It is not recognized in the environment.

#### Figure 13 Missing ruling characteristics default to docdesc

```

<envdesc envid="short-heavy-rule">
<charlist>
<ruling thick="5pt" lentype="spec" speclen="6pi" ruleclr="red"
voffset="0pt" rulepct="100" type="single">
</ruling>
</charlist>
</envdesc>
<e-i-c gi="list">
<charlist envname="short-heavy-rule">
<ruling placemnt="before">
<textbrk startln="1" endln="1">
</ruling>

```

```
<ruling placemnt="after">
<textbrk startln="1" endln="1">
...
```

The following code demonstrates that characteristic list can reference an environment and one or more characteristic subsets.

**NOTE:** The title characteristic subset is defined in **Figure 7 Useful charsubsets** on page 30. The environment `short-heavy-rule` is defined in the previous example.

#### Figure 14 Charlist references envdesc and charsubsets

```
<e-i-c gi="title" context="section">
<charlist envname="short-heavy-rule" charsubsetref="title bold">
<ruling placemnt="before">
<textbrk startln="1" endln="1">
...
```

## Document description (docdesc)

The document description value is used when no value is available for an empty characteristic and an environment is not present or does not supply a value. The document description contains a characteristic list, but a sub-characteristic list is not allowed.

### Document description example

In this example, the document description provides default values for characteristics in several categories. If these characteristics are not specified elsewhere in the FOSI, then the entire document is justified with hyphenation using the Arial font. Underscoring uses a half-point rule, with one-point rules used otherwise.

#### Figure 15 Primary formatting coded in the docdesc

```
<docdesc>
<charlist>
<font famname="Arial">
<quadding quad="justify" lastquad="relative">
<highlt scorewt="0.5pt">
<hyphen hyph="1">
<ruling thick="1pt" lentype="relative" relen="col" ruleclr="black"
type="single" voffset="0pt" rulepct="100">
</ruling>
```

**FOSI TIP** 

Use the Resolve feature in the FOSI style panels interface (covered in FOSI tutorial 1–6) to see system defaults.

**TIP** 

Probably the easiest way to understand merging and resolution is to use the Resolve feature in the FOSI style panels interface to trace each resolved characteristic to its source. This is covered in FOSI tutorial 1–6.

```
</charlist>
</docdesc>
```

## System defaults

A system default is the last resort when no other default value is available. For example, the system default for `hyphen lang` is `en`, which specifies English, and system default for `font offset` is `0pt`.

**NOTE:** System defaults are set in Arbortext Editor and cannot be altered.

## Resolving FOSI formatting specifications

The following processes are used to transform the characteristics specified in `e-i-c` into the desired formatting environment:

1. Merging characteristic subsets (`charsubsets`) into a characteristic lists (`charlists`)
2. Inheritance and defaulting
3. Merging characteristic subsets from successful attribute specifications

The result is termed the **resolved charlist**.

**NOTE:** Resolution of formatting specifications occurs at compile time, not run time.

## Merging charsubsets into charlists

FOSI's ability to specify formatting in different ways can result in more than one spec for the same characteristic. Characteristics referenced by the `charsubref` attribute on `charlist` may include some that are coded directly in the `charlist`. For example, the `charlist` may contain a `font category` with `weight="bold"` and also reference a `charsubset` that specifies `weight="medium"`. FOSI has a mechanism for determining which is used.

When the same characteristic is specified more than once, the formatter resolves them to one value in a process termed **merging**, which results in what is called the **merged charlist**. The merging process is described below.

After the merging process, the merged `charlist` may contain characteristics with no specified value, so an inheritance and defaulting process is used to

determine whether each missing characteristic inherits a value from an ancestor; defaults to a value specified in an environment, a document description default, or a system default; or has no effect at all. This is described in **Inheritance and defaulting** below. The result is called the **resolved charlist**. It defines the formatting environment for the scope of the current element.

The process used to merge charsubsets into the charlist is as follows:

1. All charsubsets referenced by the charlist's charsubsetref are merged together in the order they are listed, that is, left to right.
2. All categories specified in the charlist are merged into the result of the previous step as follows:
  - For nonrepeating categories, a characteristic specified in a referenced charsubset is used if the same characteristic is not coded directly in the charlist. When a characteristic is coded directly in the charlist, it overrides the same characteristic coded in a referenced charsubset.
  - For repeating categories, any categories coded directly in the charlist are appended to repeating categories specified in a referenced charsubset.
3. Missing categories and characteristics in the charlist are processed for inheritance and defaulting (described on page 41).
4. Any attribute rules whose conditions are satisfied are merged into the result of the previous step, as discussed in **Merging charsubsets from attspecs** on page 48.
5. The resulting charlist is processed sequentially, except for any savetext, ruling, putgraph, or usetext category with placemnt="after".

**NOTE:** The value of any variable (counter or string) that appears in a savetext conrule or usetext source is resolved at the time the savetext or usetext is processed.

Any FOSI pseudo-element that appears in a conrule is processed when the textid is processed in a usetext source. This includes the processing of the charlist for that pseudo-element.

Any pseudo-elements in a usetext source are processed in order immediately after all variables in the usetext are resolved.

See **Usetext time resolution of context** on page 476 for more information.

6. All savetext, ruling, putgraph, and usetext categories with placemnt="after" are processed in the order they are listed in the charlist.

**NOTE:** When a category has a subchars, the subchars are processed first, then the category is processed.

## Merged charlist examples

In the first example, <subtitle> is positioned flush left because quadding `quad="left" lastquad="lleft"` is coded directly in the e-i-c for <subtitle>, which overrides the center charsubset reference in the title charsubset.

### Figure 16 Charlist coding overrides charsubset reference

```
<charsubset charsubsetid="center">
<quadding inherit="1" quad="center" lastquad="lcenter">
...
<charsubset charsubsetid="title" charsubsetref="block center
keep-together keep-next bold">
...
<e-i-c gi="subtitle">
<charlist inherit="1" charsubsetref="title">
<quadding inherit="1" quad="left" lastquad="lleft">
...
```

In the next example, the e-i-c for <subtitle> calls the charsubsets title and left. Title specifies flush center. However, left is referenced last, so its formatting is used.

### Figure 17 Last charsubset reference wins

```
<charsubset charsubsetid="center">
<quadding inherit="1" quad="center" lastquad="lcenter">
...
<charsubset charsubsetid="left">
<quadding inherit="1" quad="left" lastquad="lleft">
...
<charsubset charsubsetid="title">
<font inherit="1" weight="block center keep-together
keep-next bold"
...
<e-i-c gi="subtitle">
<charlist inherit="1" charsubsetref="title left">
...
```

In this example, repeating categories coded directly in the charlist are appended to repeating categories specified in charsubsets.

## Figure 18 Repeating categories are appended to charlist

### Formatted output

first second third

### FOSI fragment

```
<charsubset charsubsetid="first">
<usetext source="first \"></usetext>
...
<charsubset charsubsetid="second">
<usetext source="\second \"></usetext>
...
<e-i-c gi="">
<charlist inherit="1" charsubsetref="first second">
<usetext source="\third\"></usetext>
...
```

## Inheritance and defaulting

It is not necessary — or desirable — to explicitly specify every characteristic in every `e-i-c` in a FOSI. Instead, FOSI has an inheritance and defaulting process that provides values for missing characteristics.

Utilizing inheritance and defaulting can make FOSI development and maintenance faster and easier. For example, inheritance makes it possible to code a single `e-i-c` in the FOSI for an inline element allowed in many contexts so it always outputs the correct formatting. For example, when the inheriting `font` category is coded in the `e-i-c` for `<bold>`, only the `weight` characteristic is specified. With inheritance enabled, missing characteristics, including `size` and `famname`, are inherited from the parent element or other ancestor.

Defaulting makes it possible to code in one place any characteristics that are the same for every use of a defaulting category. The characteristics that differ are coded in the `e-i-cs`. For example, a default rule thickness can be set for the `ruling` category to standardize the rule weight for all rules regardless of their length or positioning.

The details of inheritance and defaulting are described next, followed by **Figure 20 Inheritance and defaulting flow chart** on page 45.

**NOTE:** A simpler approach to inheritance and defaulting is described in **Chapter 3 A Practical Approach to FOSI** on page 53.

## Inheritance

The important things to understand about inheritance are:

- Inheritance applies to missing characteristics — characteristics that are not explicitly specified in an `e-i-c` — in inheriting categories.
- Inheritance can be enabled/disabled on individual inheriting categories and/or on the `charlist`.
- When inheritance is enabled, missing characteristics for an inheriting category inherit values even if the category is not present in the merged `charlist`.
- The inheritance process looks for a value in ancestor `e-i-cs`, starting with the parent. If a value is not found in any ancestor `e-i-c`, a default value is sought.
- If an inheriting category has no value for the `inherit` attribute, the value of the `charlist inherit` attribute is used.

**NOTE:** The `inherit` attribute on inheriting categories does not have a default value; it is `#IMPLIED` in the OutSpec DTD. The `inherit` attribute on `charlist`, however, has a default value of 0 (zero) in the OutSpec DTD, which specifies that inheritance is disabled.

The possible combinations are as follows:

- ▶ When `charlist inherit="1"` and `category inherit="1"`, any empty characteristics are inherited from an ancestor element, unless no value is available, in which case a default value is used.
- ▶ When `charlist inherit="0"` and `category inherit="0"`, any empty characteristics default.
- ▶ When `charlist inherit="1"` and `category inherit="0"`, any empty characteristics on the category default rather than inherit.
- ▶ When `charlist inherit="0"` and `category inherit="1"`, any empty characteristics on the category inherit.†‡

The following figure illustrates how the `charlist` and `category inherit` attributes work together to determine whether the missing characteristics in an inheriting category inherit or default.

Figure 19 Charlist and category inherit attribute matrix

		Value of category's inherit attribute		
		1	0	unspecified <sup>†</sup>
Value of charlist's inherit attribute	1	1 category inherits	0 category defaults	1 category inherits
	0	1 category inherits	0 category defaults	0 category defaults
	unspecified <sup>‡</sup>	1 category inherits	0 category defaults	0 category defaults

<sup>†</sup># IMPLIED in OutSpec DTD  
<sup>‡</sup>0 (zero) in OutSpec DTD

### Inheriting categories

- font
- leading
- highlt (highlight)
- hyphen (hyphenation)
- indent
- quadding (justification)
- sentxsp (extra sentence spacing)
- wordsp (word spacing)

## Defaulting

The important things to understand about defaulting are:

- Defaulting applies to missing characteristics — characteristics that are not explicitly specified in an `e-i-c` — in both inheriting and defaulting categories.
- A missing characteristic in a defaulting category is not assigned a default value unless that category is present in the merged charlist.
- Any missing characteristic in an inheriting category that does not receive a value from an ancestor element is assigned a default value even if the category is not present in the merged charlist.
- An empty characteristic in an inheriting category with `inherit="0"` defaults as follows:
  1. If the `e-i-c` references an environment, its value for that characteristic is used, if it exists.
  2. If an environment does not supply a default value, the document description settings are checked. If a value exists for the relevant characteristic, it is used.
  3. If no value is provided in the document description, an Arbortext-specific system default is used.
- When a defaulting category is present, defaulting is the same as listed above for inheriting categories with `inherit="0"`.

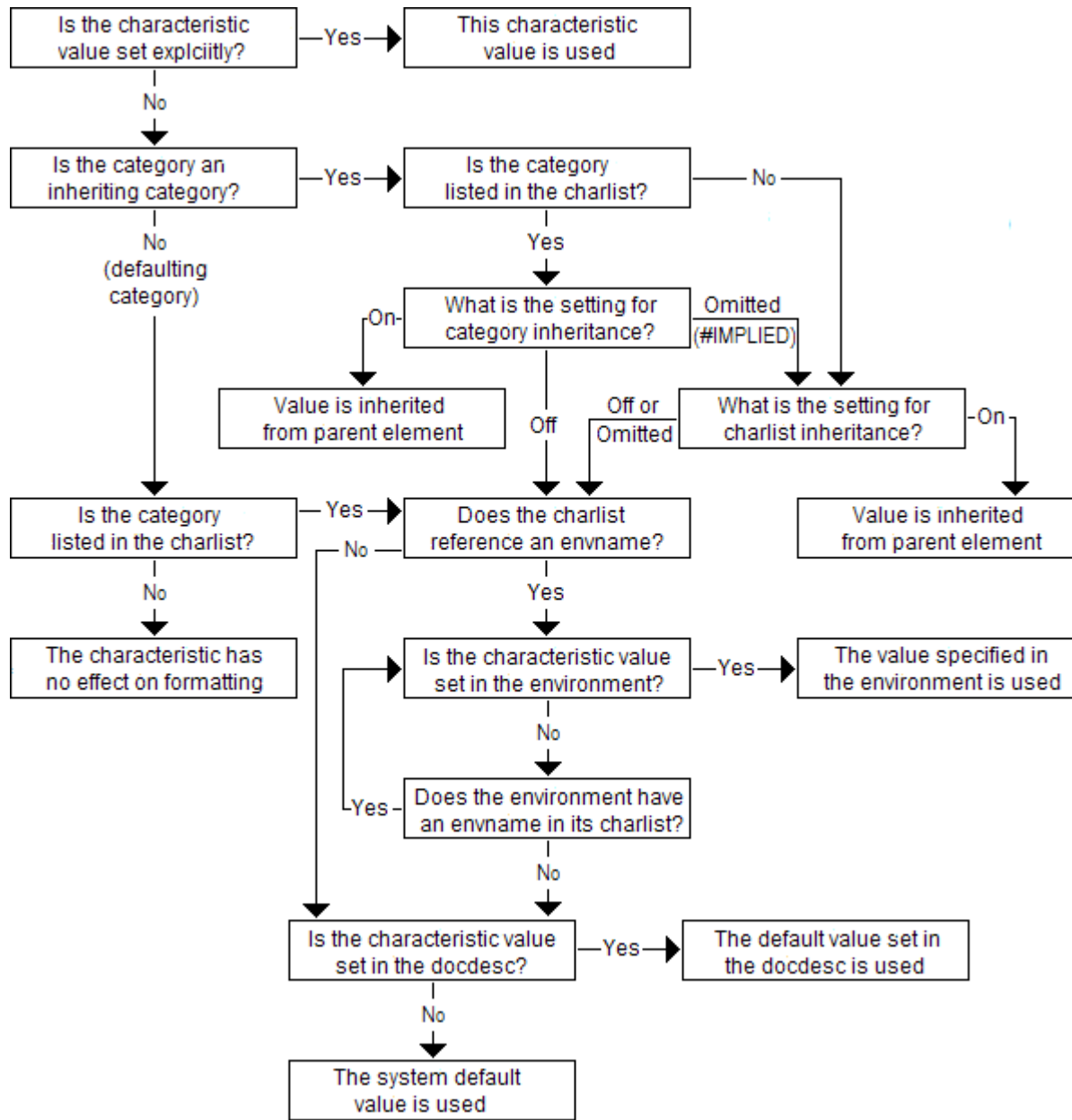
### Defaulting categories

- |                                       |                                       |
|---------------------------------------|---------------------------------------|
| ■ <code>algroup</code> (side-by-side) | ■ <code>link</code>                   |
| ■ <code>span</code>                   | ■ <code>suppress</code>               |
| ■ <code>presp</code> (prespace)       | ■ <code>reset</code>                  |
| ■ <code>postsp</code> (postspace)     | ■ <code>enumerat</code> (enumerate)   |
| ■ <code>textbrk</code> (text break)   | ■ <code>savetext</code> (save text)   |
| ■ <code>keeps</code>                  | ■ <code>usetext</code> (use text)     |
| ■ <code>boxing</code>                 | ■ <code>ruling</code>                 |
| ■ <code>chgmark</code> (change mark)  | ■ <code>putgraph</code> (put graphic) |
| ■ <code>float</code>                  |                                       |

## Inheritance and defaulting flowchart

Figure 20 below is replicated from the FOSI standard.

Figure 20 Inheritance and defaulting flow chart



## Inheritance and defaulting examples

In the first example, the `famname` characteristic is not coded directly for `<title>` and `<para>`. The `e-i-c` for `<title>` references the `title` charsubset, which specifies Century Gothic. This is the value in the merged charlist. However, the merged charlist for `<para>` does not include a value for the `famname` characteristic. The `e-i-c` for `<para>` specifies that missing font characteristics are to be inherited from the first ancestor with a value. The first ancestor in this case is `<toptag>`, which specifies Century Schoolbook.

**Figure 21** Inheriting font `famname` characteristic

### DTD fragment

```
<!ELEMENT toptag (title,para+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT para (#PCDATA)>
```

### FOSI fragment

```
<charsubset charsubsetid="title" charsubsetref="block center
title-prespace title-postspace keep-together keep-next bold">
<font inherit="1" famname="Century Gothic">
...
<e-i-c gi="para" context="toptag">
<charlist inherit="1" charsubsetref="block prespace"></charlist>
</e-i-c>
<e-i-c gi="title" context="toptag">
<charlist inherit="1" charsubsetref="title"></charlist>
</e-i-c>
<e-i-c gi="toptag">
<charlist inherit="1">
<font inherit="1" famname="Century Schoolbook">
...

```

The next example uses the DTD fragment shown in **Figure 21**. Again, the `famname` characteristic is missing from the merged charlist for `<para>`. The `e-i-c` for `<para>` specifies that missing font characteristics are to be inherited from the first ancestor with a value. However, the `e-i-c` for `<para>`'s only ancestor, `<toptag>`, does not specify a value for this characteristic. So the `docdesc` value is used.

**Figure 22** Defaulting font characteristic

```
<charsubset charsubsetid="title" charsubsetref="block center
title-prespace title-postspace keep-together keep-next bold">
<font inherit="1" famname="Century Gothic">
...

```

```

<docdesc>
<font inherit="1" famname="Century Schoolbook"
...
</docdesc>
<e-i-c gi="para" context="toptag">
<charlist inherit="1" charsubsetref="block pspace"></charlist>
</e-i-c>
<e-i-c gi="title" context="toptag">
<charlist inherit="1" charsubsetref="title"></charlist>
</e-i-c>
<e-i-c gi="toptag">
<charlist inherit="1"></charlist>
</e-i-c>

```

**Figure 23** below uses the same DTD fragment as **Figure 21** and **Figure 22**. The ruling categories coded in the `e-i-c` for `<title>` specify the `placemnt` characteristic, but other characteristics are omitted. Consequently, they default to the ruling characteristic values coded in the `docdesc`. Notice the `docdesc` value for `thick` is not used because a value for it is coded in the `e-i-c`.

**NOTE:** Categories coded within ruling in the `docdesc` have no effect. They must be coded within ruling in the `e-i-c`.

### Figure 23 Defaulting ruling characteristics

---

**RULING**

---

#### XML fragment

```
<title>RULING</title>
```

#### FOSI fragment

```

<docdesc>
...
<ruling thick="2pt" lentype="rel" relen="col" type="single">
<!-- Any categories coded here are ignored -->
</ruling>
...
<e-i-c gi="title" context="toptag">
<charlist inherit="1" charsubsetref="title">
<ruling placemnt="before">
<leading inherit="1" lead="2pt">
<textbrk startln="1" endln="1">
</ruling>
<ruling thick="4pt" placemnt="after">
<leading inherit="1" lead="8pt">
<textbrk startln="1" endln="1">
...

```

## Merging charsubsets from attspecs

After the inheritance and defaulting process, attspecs are evaluated. Charsubsets that are part of a successful attribute rule are merged in order into the merged, inherited, and defaulted charlist. A characteristic on a nonrepeating category replaces the existing characteristic. Repeating categories are appended to the merged charlist in the order they are coded in the attribute rules.

The following examples explore how this process works with repeating and nonrepeating categories.

### Merged attribute specifications examples

In the first example, the `e-i-c` specifies the default formatting for `<emphasis>` as bold. However, when the `role` attribute on `<emphasis>` specifies italic, the font category in the attribute rule is merged into the charlist. Note that `weight="medium"` is needed to override the bold charsubset referenced in the charlist.

#### Figure 24 Successful attspec overrides charlist coding

##### XML fragment

The term `<emphasis>structured markup</emphasis>` is `<emphasis role="italic">not</emphasis>...`

##### FOSI fragment

```
<e-i-c gi="emphasis">
<charlist inherit="1">
<font inherit="1" weight="bold">
</charlist>
<att>
<specval attname="role" attval="italic">
<charsubset>
<font inherit="1" posture="italic" weight="medium">
...
```

In the FOSI code above, `posture="italic"` and `weight="medium"` are specified in a font category coded in the charlist. The code below utilizes charsubsets instead of charlist coding. However, the merged formatting is the same as in the previous example.

```
<charsubset charsubsetid="bold">
<font inherit="1" weight="bold">
```

```

...
<charsubset charsubsetid="italic">
<font inherit="1" posture="italic">
...
<charsubset charsubsetid="medium">
<font inherit="1" weight="medium">
...
<e-i-c gi="emphasis">
<charlist inherit="1" charsubsetref="bold"></charlist>
<att>
<specval attname="role" attval="italic">
<charsubset charsubsetref="medium italic">
...

```

In the next example, when a reverse print (white on black) option is selected in the document, the nonrepeating `highlt` category is added to the `charlist` of the top tag in the document. Child `e-i-cs` that inherit missing `highlt` characteristics are output in reverse print.

### Figure 25 Successful `attspec` overrides `charlist` coding

#### XML fragment

```

<toptag reverse-print="yes">
...

```

#### FOSI fragment

```

<docdesc>
<highlt fontclr="#000000" bckclr="FFFFFF">
...
<e-i-c gi="toptag">
<charlist inherit="1">...</charlist>
<att>
<specval attloc="toptag" attname="reverse-print" attval="yes">
<charsubset>
<highlt inherit="1" fontclr="#FFFFFF" bckclr="000000">
...

```

In the next example, the *title* attribute on `<note>` is output from a `usetext` as the title for the `<note>`. The repeating `usetext` category is appended to `<note>`'s `charlist` following the two ruling categories, which are also repeating categories. The `placemnt` characteristics on the repeating categories control placement before or after the content of the element.

**Figure 26 Successful attspec appends repeating category****Tip**

Praesent adipiscing, ante et blandit venenatis, ante tortor venenatis lorem ante. Phasellus sceleris que tempor orci. Suspendisse ante libero. Pellentesque libero dolor, ornare noned vel velit.

**DTD fragment**

```
<!ELEMENT note ...
<!ATTLIST note title CDATA #IMPLIED>
```

**XML fragment**

```
<note title="Tip">...
```

**FOSI fragment**

```
<e-i-c gi="note">
<charlist inherit="1" charsubsetref="alert keep-together">
<ruling thick="1pt" lentype="rel" relen="col" type="single"
placemnt="before">
<leading inherit="1" lead="1pt">
<presp minimum="4pt" nominal="4pt" maximum="4pt" priority="high">
<textbrk startln="1" endln="1">
</ruling>
<ruling thick="1pt" lentype="rel" relen="col" type="single"
placemnt="after">
<leading inherit="1" lead="1pt">
<presp minimum="4pt" nominal="4pt" maximum="4pt" priority="high">
<textbrk startln="1" endln="1">
</ruling>
</charlist>
<att>
<fillval attname="title" attloc="note" fillcat="usetext" fillchar="source">
<charsubset>
<usetext placemnt="before">
<subchars charsubsetref="title"></subchars>
...
```

Appending repeating categories from successful attribute rules to existing repeating categories, as shown in the previous example, may not result in the desired order of output. Continuing the previous example, suppose the `ruling` coded in the `charlist` should come after the `<note>` *title* coded in an attribute rule but before the element content. In such cases, there are two approaches to process repeating categories in the desired order:

- Code a `usetext` — in the desired order within the `charlist` repeating categories — with `source` set to a pseudo-element that contains the `fillval` and/or `specval`. In this example, the `<note>` *title* attribute rule would be coded in the formatting pseudo-element, as shown in **Figure 27**.
- Code an unconditional `att` (one that does not contain `fillval` or `specval`) with the desired repeating category in the appropriate order among other `attspecs`. In this example, `ruling` would be coded in an unconditional `att` after the attribute rule for the `<note>` *title*. **Figure 28** provides an example of unconditional `attspecs`.

**NOTE:** In the FOSI code below, the formatting pseudo-element `<note-title.fmt>` is a child element of `<note>`, which is why the formatting pseudo-element can test or use attributes on `<note>`. Refer to **Chapter 6 FOSI Pseudo-Elements** on page 109 for details.

**Figure 27 Pseudo-element for attribute rule**

#### Tip

Præsent adipiscing, ante et blandit venenatis, ante tortor venenatis lorem ante. Phasellus sceleris que tempor orci. Suspendisse ante libero. Pellentesque libero dolor, ornare noned vel velit.

```
<e-i-c gi="note">
<charlist inherit="1" charsubsetref="alert keep-together">
<usetext source="<note-title.fmt>,</note-title.fmt>"
placemnt="before"></usetext>
<ruling thick="1pt" lentype="rel" relen="col" type="single"
placemnt="before">
<leading inherit="1" lead="1pt">
<presp minimum="4pt" nominal="4pt" maximum="4pt" condit="keep"
priority="high">
<textbrk startln="1" endln="1">
</ruling>
<ruling thick="1pt" lentype="rel" relen="col" type="single"
placemnt="after">
<leading inherit="1" lead="1pt">
<presp minimum="4pt" nominal="4pt" maximum="4pt" condit="keep"
priority="high">
<textbrk startln="1" endln="1">
</ruling>
</charlist>
</e-i-c>
<e-i-c gi="note-title.fmt">
<charlist inherit="1"></charlist>
```

```

<att>
<fillval attname="title" attloc="note" fillcat="usetext"
fillchar="source">
<charsubset>
<usetext placemnt="before">
<subchars charsubsetref="title"></subchars>
...

```

The formatted output is the same if unconditional attribute rules are used to process repeating categories in the desired order, as demonstrated in the following figure.

### Figure 28 Unconditional attribute rule

```

<e-i-c gi="note">
<charlist inherit="1" charsubsetref="alert keep-together"></charlist>
<att>
<fillval attname="title" attloc="note" fillcat="usetext"
fillchar="source">
<charsubset>
<usetext placemnt="before">
<subchars charsubsetref="title"></subchars>
</usetext>
</charsubset>
</att>
<att>
<charsubset>
<ruling thick="1pt" lentype="rel" rellen="col" type="single"
placemnt="before">
<leading inherit="1" lead="1pt">
<presp minimum="4pt" nominal="4pt" maximum="4pt" condit="keep"
priority="high">
<textbrk startln="1" endln="1">
</ruling>
<ruling thick="1pt" lentype="rel" rellen="col" type="single"
placemnt="after">
<leading inherit="1" lead="1pt">
<presp minimum="4pt" nominal="4pt" maximum="4pt" condit="keep"
priority="high">
<textbrk startln="1" endln="1">
...

```