

FOSI technical information

In this section, technical information related to FOSI formatting is presented, namely:

- **FOSI compilation** on page 10
- **FOSI structure** on page 11
- **FOSI formatting properties** on page 13
- **Element formatting** on page 16
- **Page formatting** on page 18
- **Formatting process** on page 24
- **Specifying characteristic values** on page 47
- **FOSI pseudo-elements** on page 107
- **FOSI-generated material** on page 122

FOSI compilation

FOSI is a compiled language. During FOSI development, the FOSI must be compiled in order for new coding to take effect. A FOSI is compiled automatically when a document is started in Arbortext Editor and when a different FOSI is loaded by selecting Format→Select Stylesheet (`stylesheet_select()` at the command line) or by entering the `set fosi` command at the command line.

If FOSI compilation fails, an error message is displayed and a default FOSI, `freeform.fos`, with minimal formatting is used instead. Preference settings (`set fosiwarnings` at the command line) control whether FOSI compilation messages are displayed.

There are limits to the number of `e-i-cs`, attribute rules, pagesets, and other FOSI constructs (individually and combined) supported by FOSI compilation. Please contact PTC/Arbortext Support for specifics.

TIP 

You can use the DTD Viewer to see the structure required for the FOSI. Enter **dtviewer** at the command line in the tagged FOSI editor.

FOSI structure

A FOSI is an SGML document than conforms to the OutSpec DTD, an annotated version of which is located on . A valid FOSI contains no #PCDATA, only elements and attributes. Some elements are tag pairs, including container elements; most elements in the OutSpec DTD are singletons.

The default FOSI for a document type has the same filename as the DTD plus the filename extension for FOSI, which is `.fos`. For example, `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-->
```

The next line in a `.fos` file specifies the public identifier for the OutSpec DTD, followed by any internal entity declarations, which are required for the file to be recognized as a FOSI by Arbortext Editor. 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 in a `.fos` file is an optional processing instruction that identifies the FOSI.

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

For details, see **Stylesheet ID processing instruction** on page 548.

After the top tag in the `outspec.dtd` (`<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 Lc1 _label="Adept v8.0">
```

The rest of a FOSI is organized into various **descriptions**:

- `rsrctdesc` (resource description)
- `secdesc` (security description)
- `pagedesc` (page description)
- `styldesc` (style description)

- tabdesc (table description)
- grphdesc (graphic description)
- ftndesc (footnote description)
- docdesc (document description)
- envdesc (environment description)

NOTE: Arbortext Editor provides proprietary solutions for graphics and tables and does not support FOSI `grphdesc` or `tabdesc`. Consequently, `grphdesc` and `tabdesc` are not covered in *Essential FOSI*.

Each description contains elements that specify formatting properties, as described in the next section.

NOTE: Keep in mind that FOSI is a declarative language, which defines the problem to be solved while leaving the definition of the solution to the interpreting software. FOSI describes the formatting that is desired — the “what” — while leaving “the how” to the interpreting software. In other words, FOSI does not provide a way to tell the formatting system “do this, then do that.”



FOSI formatting properties

The FOSI standard's term for what are generically called formatting properties is **characteristics**. Characteristics are arranged in groups the FOSI standard calls **categories**. Examples of FOSI categories are **font**, **ruling**, **enumerat**, and **gutter**.

Like most categories, the **font** category contains several characteristics, including a **size** characteristic that specifies the size of the type. The **gutter** category, on the other hand, has a single characteristic called **width**, which specifies the width of the gutter between columns of flowing text on a page.

NOTE: The OutSpec 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 specify the character used when the formatting system hyphenates a word.

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 attributes not characteristics. This book follows the OutSpec's lead and usually uses the term characteristic to apply to everything.

Some characteristics apply only to composed (print/PDF) output and have no effect on the Edit window display. In this book,  indicates that at least one of the category's characteristics is supported for Edit window display, while  indicates that at least one of its characteristics is supported for print/PDF output.

Characteristic values have several types, which are detailed in **Value types** on page 49.

The FOSI standard defines two kinds of characteristics: **composition characteristics** and **pagination characteristics**. The **font** category's characteristics are composition characteristics. The **gutter** category's **width** characteristic is a pagination characteristic. Composition characteristics and pagination characteristics are discussed next.

NOTE: The OutSpec DTD contains many categories and characteristics. However, it is not necessary to code them all in a FOSI. In fact, for easiest FOSI development and maintenance, it is best to code as few categories and characteristics as possible. This is feasible because the FOSI standard provides ways for formatting to be in effect without explicitly coding it. Details are in **Coding a FOSI** on page 697.

Composition characteristics

Composition characteristics apply formatting to elements in the document. It is not necessary to code every composition characteristic for every element in the document. The FOSI standard provides ways for formatting to be applied without being explicitly specified.

A composition characteristic that has no value for a particular element may inherit a value from an ancestor element. In addition, characteristic values may be called by reference to formatting macros. Also, the OutSpec DTD provides default values for some characteristics, or Arbortext Editor supplies a default. In *Essential FOSI*, FOSI code examples illustrate how to make the most of these features in order to obtain the desired formatting with as little coding as possible.

The process of supplying values for unspecified characteristics is described in **Inheritance and defaulting** on page 28.

NOTE: Some composition characteristics apply only to elements that begin a new line of text and/or end a line of text. See **Block versus inline elements** on page 37 for details.

Pagination characteristics

Pagination characteristics are applied to **layout areas** on a page. Examples of layout areas include **Column Area** and **Footnote Area**. These are detailed in **Pagedesc** on page 248 and illustrated in **Figure 1 Page Layout Areas** on page 21.

Pagination characteristics define:

- the physical appearance of pages, including size, margins, and areas for placement of text.
- how header, footer, and change mark areas appear on a page
- where floated elements may occur on a page

NOTE: Inheritance 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.

Element formatting

Element characteristics are applied to elements in the source document. Each element is matched to an **element-in-context (e-i-c)** in the FOSI, as described in **e-i-c matching** on page 25.

The formatting characteristics in an e-i-c create a formatting environment that is in effect during the scope of that element. 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.

An e-i-c specifies the name of the element to match, its **context** in the document, and its position within its parent element, which is termed **occurrence**. This means elements with the same name but different context and occurrence in the document can be formatted differently. For example, a <title> element within a <chapter> element can match one e-i-c while a <title> element within a <figure> element matches another e-i-c. Similarly, an element's occurrence within its parent element can specify different formatting. For example, the first <paragraph> after a <title> may have no indent while all other <paragraph>s within the parent element are indented.

Both context and occurrence can be specified to provide different formatting, as illustrated in the following examples.

- e-i-c gi="para" context="chapter" occur="first"
- e-i-c gi="para" context="chapter" occur="notfirst"
- e-i-c gi="item" context="list" occur="only"
- e-i-c gi="item" context="list" occur="first"
- e-i-c gi="item" context="list" occur="middle"
- e-i-c gi="item" context="list" 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. If any e-i-c's in a FOSI use XPath, the rsrcdesc attribute eicorderispriority must be set to "1", which

FOSI TIP

There are other ways to select a specific element if occurrence is not sufficient. For example, the FOSI can be coded to count elements and apply formatting according to the element count.

means that the e-i-c's for an element are evaluated in order of occurrence in the FOSI, and the first one that matches is used.

An e-i-c consists of two parts: **charlist** and **attspecs**, also known as **attribute rules**.

The **charlist** contains categories in the order defined in the OutSpec DTD. Categories such as **font** are allowed just once for an e-i-c because text can only be shown in one font. These categories are termed **nonrepeating categories**. Categories such as **ruling**, on the other hand, may be specified in an e-i-c as many times as needed. These categories are called **repeating categories**.

An attribute rule may contain categories that use an attribute value as a characteristic value or as document content. An attribute rule may also contain categories that test the value of an attribute and, depending on the outcome of the test, add the designated nonrepeating and repeating categories to the e-i-c's formatting environment.

When an e-i-c is formatted, its content (if any) is added to the flowing text. 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 further in the next section.

Page formatting

As described in the previous section, element content is formatted by `e-i-cs` and becomes flowing text that fills columns and pages. Each page has 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 may have a single column while the pages in the body of the document contain two columns of flowing text. In addition, landscape pages may be needed for wide tables and graphics, and foldout pages may be needed for oversize graphics.

Additional examples of differing page layouts include:

- 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
- blank pages are totally blank, without the page headers and footers that appear on pages with content

Pagesets and page models

In a FOSI, one or more **pagesets** with one or more **page models** are used to handle the necessary 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

TRIVIA

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.

- 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

Calculated column widths and depths

The formatting engine uses the values specified for the above-listed formatting properties 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 a wider left margin on recto pages and a wider 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`.

Blank pages

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 a blank page.

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

Page model references

FOSI supports references to an existing page model. For instance, a blank page may be defined once and referenced in one or more 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 the numbering, resetting the numbering, and supporting references to page numbers.

Page Layout Areas

Each page model contains Page Layout Areas, as shown in **Figure 1** on the next page, and Float Areas, as illustrated in **Figure 2** on the page after that. The shaded areas rotate for landscape pages.

Figure 1 Page Layout Areas

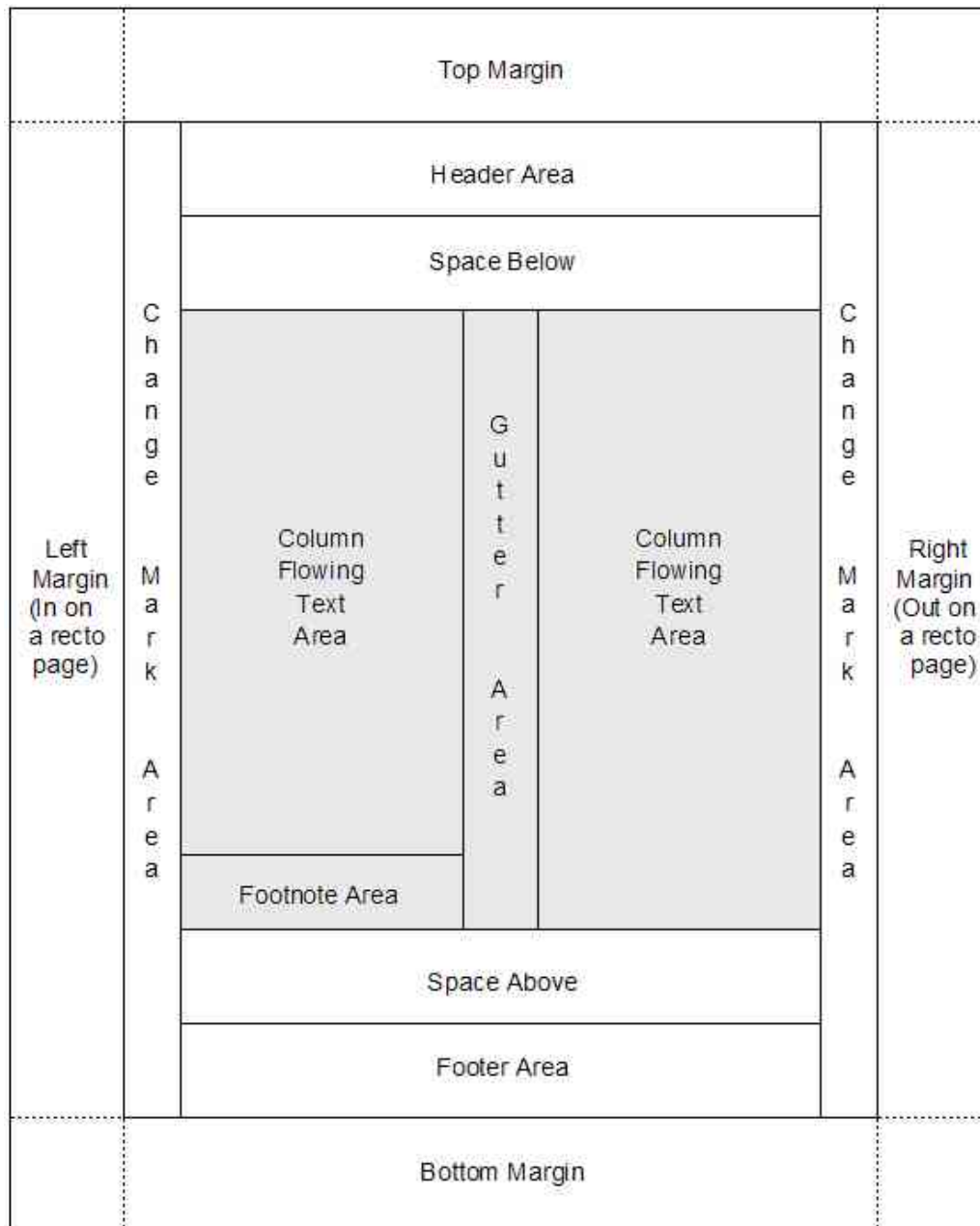
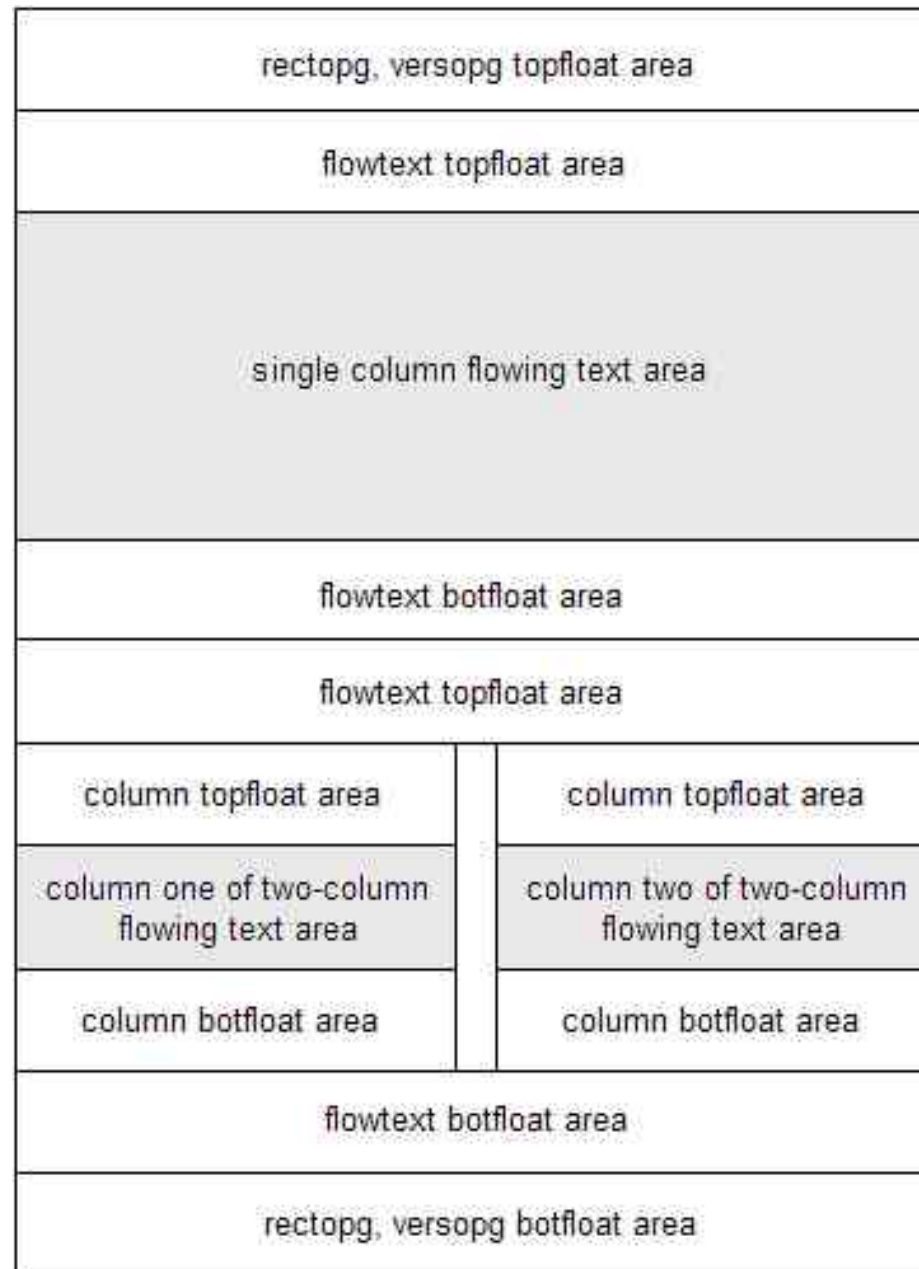


Figure 2 Float Areas



Page fidelity versus 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.

The FOSI standard does not provide enough control to support page fidelity or page integrity. While it 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:

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

Consequently, different formatters produce different results. Even a single formatting engine evolving over many software releases will produce 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.