Galleys

by Peter B. West

Table of contents

1. Layout galleys in FOP

1.1. Galleys in Lout

Jeffrey H. Kingston, in *The Design and Implementation of the Lout Document Formatting Language* Section 5 (http://snark.niif.spb.su/~uwe/lout/design.pdf) , describes the **galley** abstraction which he implemented in *Lout*. A document to be formatted is a stream of text and symbols, some of which are **receptive symbols**. The output file is the first receptive symbol; the formatting document is the first galley. The archetypical example of a receptive symbol is **@FootPlace** and its corresponding galley definition, **@FootNote**.

Each galley should be thought of as a concurrent process, and each is associated with a semaphore (or synchronisation object.) Galleys are free to "promote" components into receptive targets as long as

- an appropriate target has been encountered in the file,
- the component being promoted contains no unresolved galley targets itself, and
- there is sufficient room for the galley component at the target.

If these conditions are not met, the galley blocks on its semaphore. When conditions change so that further progress may be possible, the semaphore is signalled. Note that the galleys are a hierarchy, and that the processing and promotion of galley contents happens *bottom-up*.

1.2. Some features of galleys

It is essential to note that galleys are self-managing; they are effectively layout *bots* which require only a receptive area. If a galley fills a receptive area (say, at the completion of a page), the galley will wait on its semaphore, and will remain stalled until a new receptive area is uncovered in the continued processing (say, as the filled page is flushed to output and a new empty page is generated.)

Difficulties with this approach become evident when there are mutual dependencies between receptive areas which require negotiation between the respective galleys, and, in some cases, arbitrary deadlock breaking when there is no clear-cut resolution to conflicting demands. Footnote processing and side floats are examples. A thornier example is table column layout in *auto* mode, where the column widths are determined by the contents. In implementing galleys in FOP, these difficulties must be taken into account, and some solutions proposed.

Galleys model the whole of the process of creating the final formatted output; the document as a whole is regarded as a galley which flushes in to the output file.

1.3. The layout tree

This proposal for implementing galleys in FOP makes use of a **layout tree**. As with the layout managers (http://xml.apache.org/fop/design/layout.html) already proposed, the layout tree acts as a bridge between the FO Tree (http://xml.apache.org/fop/design/fotree.html) and the Area Tree (http://xml.apache.org/fop/design/areas.html) . If the elements of the FO Tree are FO nodes, and the elements of the Area Tree are Area nodes, representing areas to be drawn on the output medium, the elements of the layout tree are **galley nodes** and **area tree fragments**. The area tree fragments are the final stages of the resolution of the galleys; the output of the galleys will be inserted directly into the Area Tree. The tree structure makes it clear that the whole of the formatting process in FOP, under this model, is a hierarchical series of galleys. The dynamic data comes from fo:flow and fo:static-content, and the higher-level receptive areas are derived from the *layout-master-set*.

1.4. Processing galleys

Galleys are processed in two basic processing environments:

1.4.1. Inline- and block-progression dimensions known

The galley at set-up is provided with both an *inline-progression-dimension* (*i-p-d*) and a *block-progression-dimension* (*b-p-d*). In this case, no further intervention is necessary to lay out the galley. The galley has the possibility of laying itself out, creating all necessary area nodes. This does not preclude the possibility that some children of this galley will not be able to be so directly laid out, and will fall into the second category.

While the option of "automatic" layout exists, to use such a method would relinquish the possibility of monitoring the results of such layout and performing fine-tuning.

1.4.2. Inline- ior block-progression-dimensions unknown

The galley cannot immediately be provided with an i-p-d ior a b-p-d. This will occur in some of the difficult cases mentioned earlier. In these cases, the parent galley acts as a layout manager, similar to the sense used in another discussion (http://xml.apache.org/fop/design/layout.html) . The children, lacking full receptive area dimensions, will proceed with galley pre-processing, a procedure which will, of necessity, be followed recursively by all of its children down to the atomic elements of the galley. These atomic elements are the individual *fo:character* nodes and images of fixed dimensions.

1.5. Galley pre-processing

Galley pre-processing involves the spatial resolution of objects from the flows to the greatest extent possible without information on the dimensions of the target area. Line-areas have a block progression dimension which is determined by their contents. To achieve full generality in layouts of indeterminate dimensions, the contents of line-areas should be laid out as though their inline progression dimension were limited only by their content. In terms of inline-areas, galleys would process text and resolve the dimensions of included images. Text would be collected into runs with the same alignment characteristics. In the process, all possible "natural" and hyphenation break-points can be determined. Where a line-area contains mixed fonts or embedded images, the b-p-d of the individual line-areas which are eventually stacked will, in general, depend on the line break points, but the advantage of this approach is that such actual selections can be backed out and new break points selected with a minimum of re-calculation. This can potentially occur whenever a first attempt at page layout is backed out.

Figure 1

Galley pre-processing diagram

Once this pre-processing has been achieved, it is envisaged that a layout manager might make requests to the galley of its ability to fill an area of a given inline-progression-dimension. A positive response would be accompanied by the block-progression-dimension. The other possibilities are a partial fill, which would also require b-p-d data, and a failure due to insufficient i-p-d, in which case the minimum i-p-d requirement would be returned. Note that decisions about the actual dimensions of line-areas to be filled can be deferred until all options have been tested.

The other primary form of information provided by a pre-processed galley is its minimum and maximum i-p-d, so that decisions can be made by the parent on the spacing of table columns. Apart from information requests, higher-level processes can either make requests of the galleys for chunks of nominated sizes, or simply provide the galley with an i-p-d and b-p-d, which will trigger the flushing of the galley components into Area nodes. Until they have flushed, the galleys must be able to respond to a sequence of information requests, more or less in the manner of a request iterator, and separately manage the flushing of objects into the area tree. The purpose of the "request iterator" would be to support "incremental" information requests like *getNextBreakPosition*.