This paper describes the architecture of an experimental document composition system named JANUS, which is intended to support authors of complex documents containing mixtures of text and images. The JANUS system is highly interactive, providing authors with immediate feedback and direct electronic control over page layouts, using a special two-display workstation. Authors communicate with the system by marking up their documents with high-level descriptive "tags." A tag definition language is provided whereby new tags may be defined and the format of each tagged object may be controlled.

JANUS: An interactive document formatter based on declarative tags

by D. D. Chamberlin, O. P. Bertrand, M. J. Goodfellow, J. C. King, D. R. Slutz, S. J. P. Todd, and B. W. Wade

JANUS is the name of an experimental document composition system intended to provide interactive support for authors of complex documents containing mixtures of text, line art, and tone art. Typical examples of such documents are technical manuals for assembly, maintenance, and repair of equipment, which may have several illustrations on each page. In today's technology, production of these documents typically involves a manual "pasteup" step in which illustrations are merged with text and individual page layouts are determined. Manual pasteup is labor-intensive and time-consuming, and it results in a substantial delay from the time the written text is complete to the time that the camera-ready copy is prepared. Manual pasteup also greatly increases the difficulty and expense of making revisions to a document, or of maintaining multiple versions of a document (for various models of a device, for example). One of the objectives of JANUS is to provide an interactive page layout capability, thus permitting the author to control placement of illustrations and text electronically and to view the finished pages immediately on a graphic display. This objective has been made feasible by the continuing decline in the cost of digital storage and processing, and by the advent of inexpensive high-function displays and printers which are capable of displaying and printing images as well as text in various fonts. This paper describes the architecture of JANUS.

classifying document composition systems

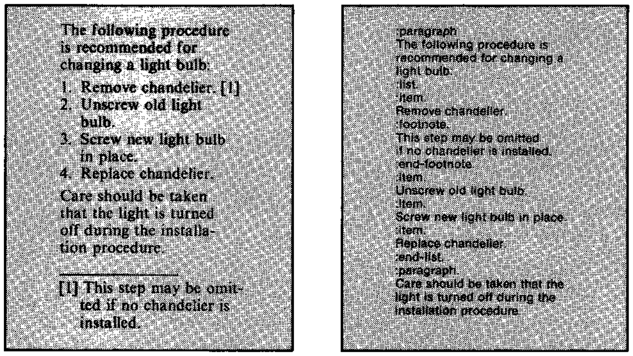
In order to define a background for the architecture of the JANUS system, we will introduce three ways of classifying document composition systems. These three classifications may be thought of as orthogonal axes that define a three-dimensional space in which each document system is represented by a point.

Batch versus interactive. The first classification distinguishes systems that view document formatting as a "batch" job from those that are interactive. Batch-oriented systems, such as IBM's Document Composition Facility (also known as SCRIPT/VS),[1] Donald Knuth's TEX,[2] and Brian Reid's SCRIBE,[3] begin at the first page of a document and proceed to the last page, transforming an input file of text and markup commands into a formatted output file. This process takes place without the participation of the author, and the effect of a local change in the document can be seen only by reformatting the entire document. Interactive systems such as IBM's Displaywriter[4] or the Xerox STAR[5] permit the author to view and edit the formatted document directly and to see the effects of his changes immediately in their local context. Interactive systems combine the traditionally separate functions of "editor" and "formatter" into a single system so that authors can interact with both functions without changing environments.

Text only versus images, graphics, and text. Our second classification distinguishes systems that process only text (possibly including multiple fonts) from systems that process images and graphics as well. A full-function system of the latter kind will include on-line digital storage for both line art (black and white images and graphics) and tone art (gray-scale or half-tone images). It should be possible to display all these types of information at the author's workstation, and to print them on the same medium as the text (either an all-points-addressable printer of adequate resolution or a photocomposer). The advantages of on-line storage of illustrations are obvious. The author can be given direct electronic control over the final appearance of the printed page, thus avoiding the expensive and time-consuming manual pasteup step. Perhaps even more important, when the entire document is in digital form, it can be communicated electronically from one location to another; it can be archived on magnetic storage media; multiple versions can be maintained under computer control; and the document can be printed on demand in "customized" versions for different users or purposes. Commercial systems offering many of these advantages include the "AIDS" system of Information International, Inc.,[6] and the "Response 300" system of Scitex.[7] The SCRIBE system at Carnegie-Mellon University[3] also has a capability for imbedding digitized images in documents.

Procedural versus declarative. Our third classification is based on the level of the commands that the author uses to mark up his document and the degree to which these commands describe the structure of the document. In a "procedural" system, the author

Figure 1 Example of a marked-up document



The following procedure is recommended for changing a light bulb:

1. Remove chandelier. [1]
2. Unscrew old light bulb.
3. Screw new light bulb in place.
4. Replace chandelier.

Care should be taken that the light is turned off during the installation procedure.

[1] This step may be omitted if no chandelier is installed.

Formatted document

```
:paragraph
The following procedure is
recommended for changing a
light bulb:
:list.
:item.
Remove chandelier.
:footnote.
This step may be omitted
if no chandelier is installed.
:end-footnote.
:item.
Unscrew old light bulb.
:item.
Screw new light bulb in place.
:item.
Replace chandelier.
:end-list.
:paragraph.
Care should be taken that the
light is turned off during the
installation procedure.
```

Original markup

controls formatting by means of low-level commands which direct specific actions, such as "skip two lines," "enter italic font," or "indent one inch." In such a system, the author specifies the desired output, but gives no hint as to the *reason* for the command. A switch to italics may be executed to emphasize a phrase, or to display the title of a book, or to set off a section heading—in each case, the command is the same. Recently, a few systems that allow an author to mark up his document using a higher level of notation have become available; such systems simultaneously ease the markup task and provide more assistance to the author in accomplishing his purpose. Examples of such systems, which we will refer to as "declarative" systems, are IBM's GML[8] and Brian Reid's SCRIBE.[3] In a declarative system, an author marks up his document with "tags" that identify the various parts of the document, such as chapter headings, numbered lists, and footnotes. Each tagged item is formatted according to the instructions contained in a procedure specific to that tag. Declarative systems provide authors with the benefits of a high-level markup language in which complex formatting procedures can be invoked by simple tags; they also provide uniformity of style across documents, since the appearance of a footnote, for example, is controlled by the tag procedure rather than by individual authors. In addition, declarative systems make marked-up documents independent of any specific output device; for example, a "book title" tag might result in italics on one output device and in underlining on another.

An example document that illustrates the advantages of a declarative system is shown in Figure 1. We show both the formatted document and the "markup" from which it was derived. The document contains a numbered list and a footnote. In a procedural system, the author of this document would need to number the list items himself and would control the formatting (spacing, indentation, placement on the page, etc.) of the list items and footnote by dozens of low-level commands. In a declarative system the author simply identifies the parts of the document by means of tags such as the ":item" and ":footnote" tags illustrated in Figure 1 (the leading colon is used to distinguish tags from text). The simplicity and convenience of a declarative system is further demonstrated when we consider the process of editing a document. If the author needs to add a new item to the top of the list, a procedural system would require him to manually renumber all the list items. In a declarative system, this problem is solved automatically when the author inserts a new list item identified by the ":item" tag.

An important problem facing the designer of a declarative document composition system is that of providing an interface by which tags, and the formatting actions required by these tags, are described to the system. Such an interface must provide a means for defining new tags and for modifying the definitions of existing tags. This tag-definition interface may or may not be made available to individual authors, according to the editorial policy of the organization. It is our opinion that the usability of the tag-definition interface is critical in the success of any declarative formatting system. Details of the JANUS approach to defining tags will be given in a later section.

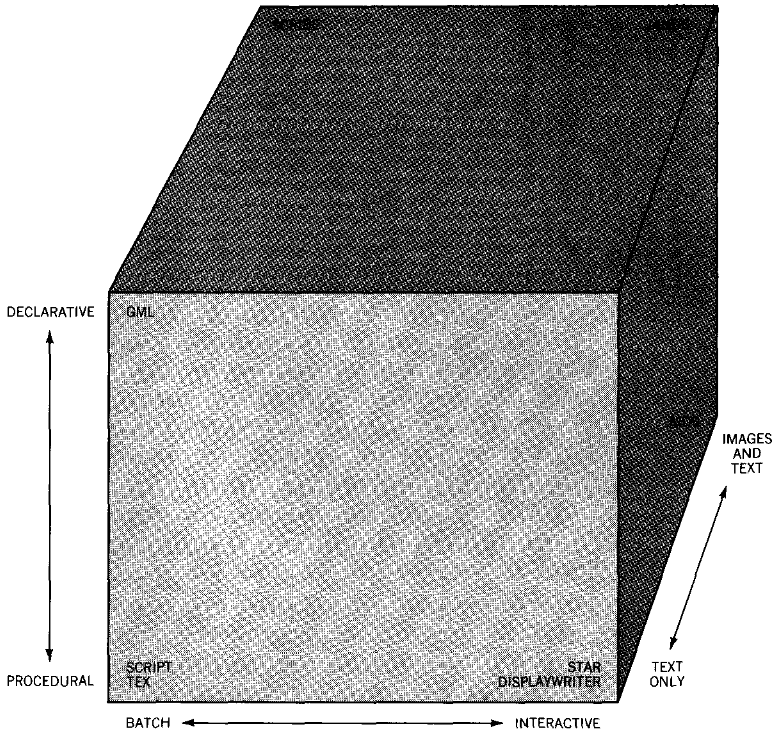
The three classifications, together with some of the document systems mentioned above, are represented as a three-dimensional diagram in Figure 2. It is the objective of the JANUS project to build a system that is interactive, declarative, and capable of processing images as well as text.

This paper will describe the hardware environment of the experimental JANUS prototype as well as the two languages implemented by this prototype: (1) a markup language for imbedding descriptive tags in a document, and (2) a language whereby new tags and document types may be defined. The paper will then describe the JANUS software architecture, focusing particularly on techniques that permit the JANUS user to skip from one place to another in a document, seeing the effects of his/her editing changes without reformatting the document from the beginning.

The JANUS workstation

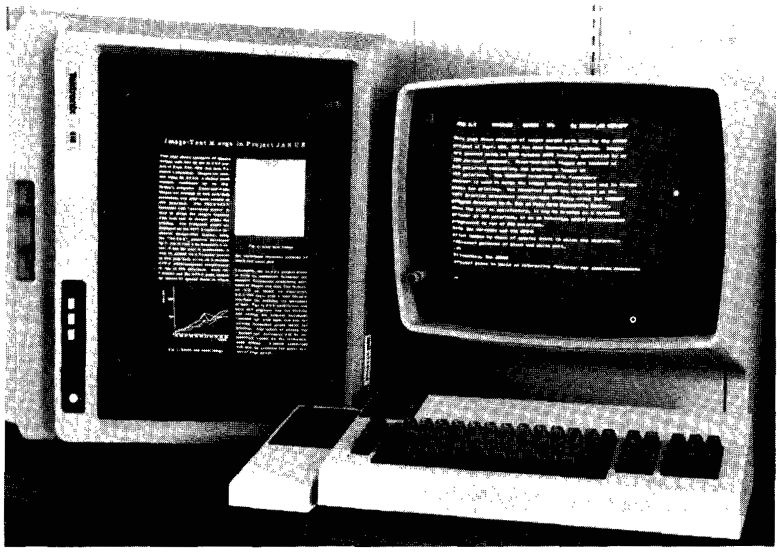
Choice of a workstation for the JANUS system was dictated by the objectives discussed above. To give a good interactive representation of the formatted page, the workstation must include an all-points-addressable display of adequate size and resolution for displaying full-size pages of images, graphics, and text. The declarative tags that describe document structure should be visible also, since it is by entry and modification of these tags that the author gives the system the information it needs to format the document. One approach that was considered was to somehow overlay the tags on the display of the formatted page, using a different color or some other means to

Figure 2 Classifications of document systems



distinguish tags from text. However, we feel that this would lead to a confusing display and would place unacceptable constraints on the space available for display of tags. Furthermore, the author's markup may encompass several versions of the document, whereas a given formatted page can represent only one of these versions. For these reasons, the JANUS project adopted a "two-display" approach, in which the original "marked-up" document and the final formatted document are displayed side by side, with the same portion of the document visible simultaneously on the two displays. As the author edits the text and tags visible on the "markup" display, he may invoke a command to see the effects of his actions on the final document in the "formatted page" display. As the author moves from one place to another in the markup file, the formatted page display tracks his position in the final document. The two-display workstation suggested the name for our project, which is named after the two-faced Roman god JANUS. Although the JANUS prototype uses two separate display screens, it would also be possible to combine the two JANUS displays on a single high-function screen either by space-multiplexing (splitting the screen) or time-multiplexing (allowing the user to toggle between the markup display and the formatted page display).

Figure 3 JANUS workstation



The two-screen workstation selected for use in the JANUS project is the IBM 3277 Graphics Attachment.[9] This workstation consists of an IBM 3277 display terminal, which provides a 24-line CRT (cathode-ray tube) on which the markup file may be displayed and edited, and a Tektronix 618 nineteen-inch direct-view storage tube, which provides a full-page-size, all-points-addressable screen for display of the formatted document. The workstation also provides a joystick which can be used for "pointing" to specific positions on the storage-tube display, a necessity for some of the interactive commands to be described later. The JANUS workstation is illustrated in Figure 3.

An experimental prototype of the JANUS system is currently being implemented to run on a System/370 under control of the Virtual Machine/Conversational Monitor System (VM/CMS) operating system. Our source of images is an ECRM Autokon 8400 scanner, controlled by an IBM Series/1 computer, which buffers scanned images and forwards them to the System/370 via a teleprocessing link. The JANUS system will also accept graphic input from various graphic editors such as PANEL2.[10] The formatted documents may be directed to a variety of output devices, including an Autologic APS-5 photocomposer.

The JANUS markup language

The notation made available to JANUS users for marking up their documents is a variation of the "Generalized Markup Language" (GML) implemented by IBM's Document Composition Facility.[8]

In a JANUS document, as in GML, each structural part of the document is identified by a "tag" which begins with a special delimiter (by default, a colon ":"). Associated with each tag is a "scope" which consists of that part of the document that is described by the tag. Inside the scope of a tag may be text and other tags (e.g., a list may contain items, which may in turn contain paragraphs). Any tag may be used in either a "short form" or a "long form," each of which is described below.

short form

In the short form, a tag is immediately followed by a delimiter, and its scope is terminated by a matching delimiter. For example, suppose that the ":Q" tag identifies a quotation. Its short form is as follows:

```
:Q/To be or not to be; that is the question./
```

Any nonalphameric delimiter (other than a period) may be used to enclose the scope of a tag in short form. The following example illustrates a short-form ":Q" (quotation) tag that contains within its scope another short-form ":HP" (highlighted phrase) tag:

```
:Q/To be or not to be; :HP!that! is the question./
```

long form

A long-form tag is delimited by a period (.); its scope extends from the period until it is ended by one of the following rules:

Rule 1: The scope of any tag may be terminated by a matching "ending" tag whose name has a leading "e." The following example illustrates the long form of the ":Q" (quotation) tag:

```
:Q.
To be or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
And by opposing end them.
:EQ.
```

Rule 2: The scope of a tag is automatically terminated when a tag that includes it is terminated.

Rule 3: The scope of any tag is automatically terminated when another tag is encountered that cannot be directly contained in it. The definer of each tag must provide a list of other tags that it may directly contain. The functioning of this rule is illustrated by an example. Suppose the following tags and nesting rules have been defined:

:P(Paragraph) may directly contain :LIST

:LIST may directly contain :ITEM

:ITEM may directly contain :P or :LIST

The following example represents valid usage of these tags. The text accompanying the tags indicates how the scope of each tag is recognized by the JANUS parser.

```
:P.This tag begins a big paragraph. As you will see, this paragraph
contains two levels of nested lists.
:LIST.
:ITEM.This is item 1 of the outer list.
:ITEM.This is item 2. It automatically terminates item 1 because an
item cannot contain another item. However, an item can contain a
list, and this one does.
:LIST.
:ITEM.This is item 2a of the inner list which is inside item 2.
:ITEM.This is item 2b of the inner list.
:ELIST.
:P.The "ELIST" tag terminated the inner list, but we are still inside
item 2. This is a new small paragraph inside item 2.
:ELIST.
Text occurring here is outside the lists but it is still part of the
original big paragraph.
:P.This tag begins another big paragraph and terminates the
original one, because a paragraph cannot directly contain another
paragraph.
```

A tag may be defined so as to accept certain parameters in addition to its scope. For example, a level-one heading tag may accept an "ID" parameter which is used to refer to the heading from other places in the document. This option is expressed in JANUS markup in the same way as in GML: by a list of parameter names and values, immediately preceding the scope of the tag. The following example illustrates an "ID" parameter for a level-one heading:

```
:H1 ID= 'bike'/How to Ride a Bicycle/
```

The principal differences between the JANUS markup language and IBM's General Markup Language (GML) are as follows:

1. Both GML and JANUS have a "short form" and a "long form" for tags. However, in GML, each tag can take only one of these forms. In JANUS markup, any tag can be used in either the short form or the long form.
2. In the current GML implementation, short-form tags are delimited by end-of-line and do not permit other tags to occur within their scope. In JANUS markup, short-form tags are explicitly delimited and may contain other tags within their scope. This permits useful cases such as a highlighted phrase or a Greek letter within a chapter heading.

Defining new types of documents

As described in the previous section, JANUS permits the user to mark up his document with high-level descriptive tags that identify various parts of the document. For this approach to be effective, a means must be provided whereby new tags can be defined and the system can be given knowledge about the relationships among the tags and the effect of each tag on formatting. All the information that enables the system to recognize and process the tags for a particular type of document is called a "document profile."

We believe that providing an effective means for defining document profiles is the central problem in implementation of a declarative document system. Various systems have approached this problem in different ways. IBM's GML product[8] defines each tag by an "application processing function" written as a macro based on low-level formatting commands. The SCRIBE formatter[3] defines each tag by its effect on an "environment" vector that controls fonts and justification and contains counters for lists, etc.

The JANUS approach to defining document profiles was influenced by the following observations:

- Page formatting is a two-dimensional problem, similar in some ways to the well-known "bin-packing" problem of packing objects of various sizes into a fixed space. Therefore, a two-dimensional graphic interface would be helpful for describing page layouts.
- Definition of a new type of document takes place much less frequently than creation of one specific document. Therefore, it is reasonable to expect a certain degree of sophistication from the designer of a document type. However, the language provided for this purpose should be consistent, well-structured, and readable.
- Wherever possible, the definition of a tag should be independent of other tags that may occur in the same document. For example, a paragraph may occur inside a numbered list, a footnote, an abstract, etc.; hence, the tag definition for a paragraph should ideally be independent of the environment in which it is used.

In JANUS, the process of defining a document profile consists of three parts:

1. The user lists the tags to be used in the new document type. For each tag, the user specifies its parameters, defaults, and nesting rules (i.e., which other tags may be directly nested inside this tag). This list of tags and rules completely specifies the structure of the document type.
2. For each tag, the user writes a tag routine that specifies how the tag formats its scope into a "galley"—a long column of text similar to the galleys used in publishing.
3. The user specifies a set of "page templates" that control how the galleys are electronically cut and packed onto pages.

The concept of a "galley" cleanly separates the text justification (column-forming) process from the page makeup process and allows each of these processes to be controlled by a specialized language. The languages for controlling galley formation and page makeup are now described.

Tag routines

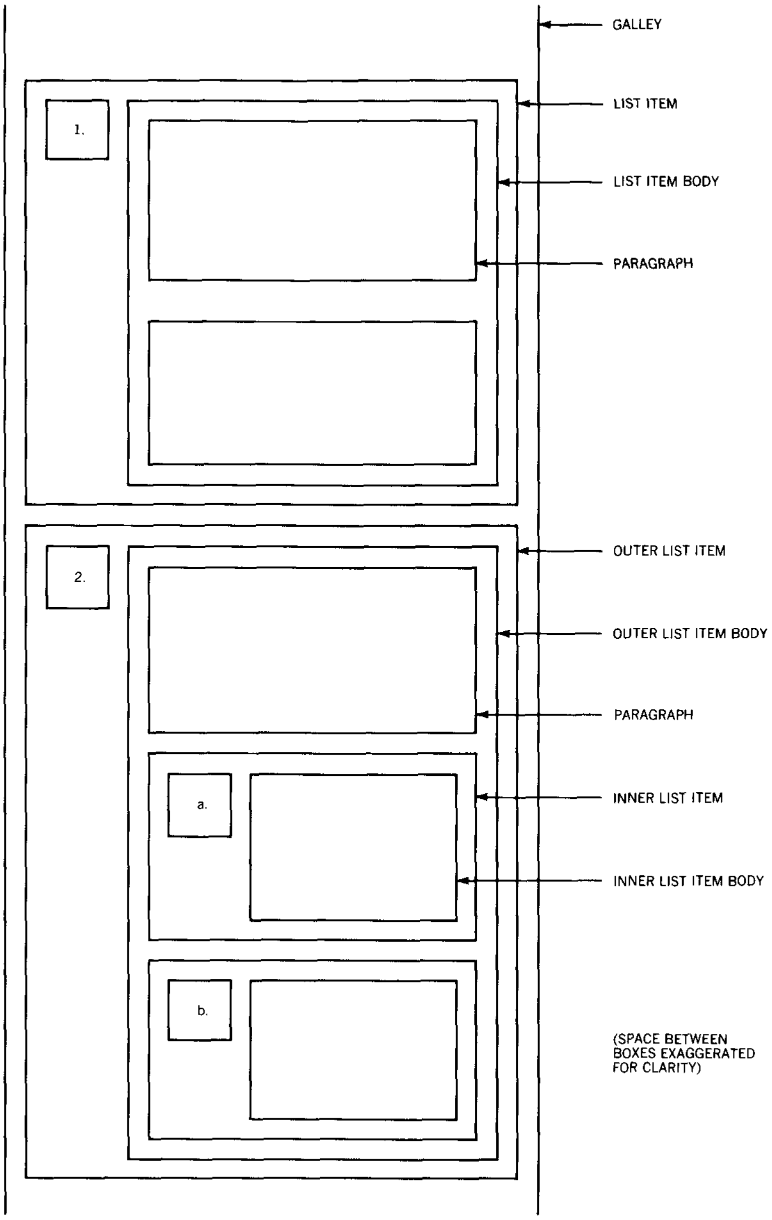
The JANUS language for writing tag routines borrows a concept from the TEX system of Donald Knuth:[2] the idea that pages are made up of *boxes* of various sizes, shapes, and properties. Each tag routine creates the boxes it needs for the desired formatting and fills them with the text found within its scope. If, during the process of filling a box, another tag is encountered, its tag routine is called as a subroutine, and it may create additional boxes inside the "parent" box. Since a document is a collection of nested tags, the tag routines will produce a collection of nested boxes called a "proto-galley," as shown in Figure 4. Figure 4 illustrates the structure of the proto-galley that might result from a list containing two items. The first item consists of two paragraphs. The second item consists of one paragraph and a sublist which in turn contains two items. The boxes that contain the inner list were created by recursive invocations of the tag routines for the "list" and "item" tags.

The language provided in the JANUS prototype for writing tag routines is Pascal, a high-level structured language originally proposed by Niklaus Wirth as an introductory student programming language.[11] JANUS tag routines, as well as the JANUS prototype itself, are written in IBM's Pascal/VS,[12] which includes several extensions to the original Pascal language, including varying-length character strings. The JANUS system makes available certain specialized procedures that may be called by the Pascal tag routines. The most important of these are described as follows:

- A BOX command is used by the tag routine to create a new box. The size and position of a box are always specified in relation to the "parent" box. In this way, the tag is isolated from its surroundings; e.g., a paragraph tag need not know whether it occurs inside several levels of nested lists. In addition to its size and shape, a box has the following properties:

  1. *HINGES:* Each box has a "top hinge" and a "bottom hinge" which specify how closely this box may approach other boxes in the vertical direction. If a box containing a chapter heading has a "top hinge" of one inch, this means that when pages are made up, no other box will be placed within one inch of the top of the heading box.
  2. *SHIELDS:* In general, the galley can be broken between any two lines of text for the purposes of page makeup. However, a box may "shield" part or all of its contents to prevent it from being separated by page or column breaks. For example, a

Figure 4 A proto-galley



"paragraph" box might shield its first two and last two lines to prevent "widows." A box containing a table might shield its entire contents, ensuring that the table is not broken across two pages.

3. *PLACEMENT:* When pages are made up from the galley, most boxes are placed on the page in sequential order. However, a tag routine can create two kinds of special boxes: "floating" boxes and "fixtures." A "floating" box is allowed to float out

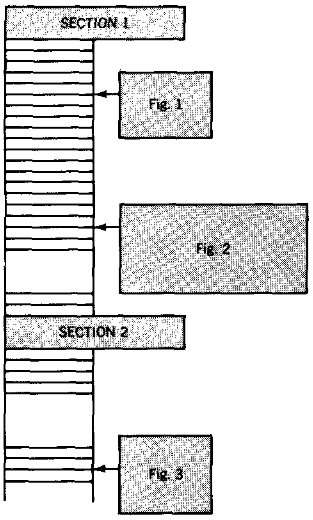
of its original order in the galley to enable more attractive page makeup (e.g., a figure). A "fixture" is a box that is repetitively placed in the same position on each page (e.g., a running title).

- A STATE command is used by the tag routine to control the process of justifying text and placing it in boxes. The STATE command can specify fonts and cause text to be justified, centered, ragged-right, etc. Each STATE command applies only to the box that is currently being filled and to its descendants. The system maintains a stack of STATE settings for all active boxes, and reverts to an earlier STATE whenever a tag terminates and closes its box. This is another measure that simplifies tag writing by isolating tags from one another.
- A JANPARSE command is provided, which instructs the system to fill the current box with text from the source document, calling other tag routines as tags are encountered, until the end of the scope of the current tag.

The Appendix contains two sample tag routines which implement simple tags called LIST and ITEM. Both of these tag routines are recursive, and in case of a list inside another list they might produce the proto-galley shown in Figure 4.[13]

The "proto-galley" produced by the running tag routines is automatically converted by the JANUS system into a "galley." In the galley, the nested boxes are no longer visible. The galley consists of a series of discrete, indivisible *slugs* (the term "slug" is derived from the printer's term for a piece of metal type). Each line of text in the proto-galley becomes a separate slug unless it is joined to another line by a shield. Each galley has two widths associated with it: a *column width* and a *page width* (if the galley is to be used for single-column formatting, the two widths are the same). Each slug takes on one of these two widths, and inherits hinges and placement attributes (SEQUENTIAL, FLOATING, or FIXTURE) from the box(es) from which it was derived. The appearance of a galley after it has been organized into slugs is shown in Figure 5.

Figure 5 A galley



New tag routines are added to the JANUS system by compiling them and link-editing them to the JANUS formatter. In this way, the power and ease of use of a high-level language are combined with the efficiency of compiled code. We believe that the JANUS approach simplifies the task of writing tag routines for the following reasons:

1. The tag routine need not specify how the tag is terminated (or whether it terminates other tags); this process is controlled automatically by the JANUS parser, based on rules declared outside the tag routines.
2. The tag routine is not concerned with pagination; this is controlled by "page templates" which are specified outside the tag routines.

3. The tag routine is made independent of other tags by the concept of "boxes"—each routine works inside the box inherited from its parent.
4. The tag writer has available the great power and simplicity of a modern, well-structured programming language, Pascal.

Page templates

As described above, the result of the tag routines operating on an input document is one or more "galleys" of formatted text. In general, a document may have several galleys. For example, one galley may contain the main body of text, another may contain footnotes, and a third may contain a list of bibliographic references. The tag routines may direct their output to specific galleys by issuing commands as they process the document.

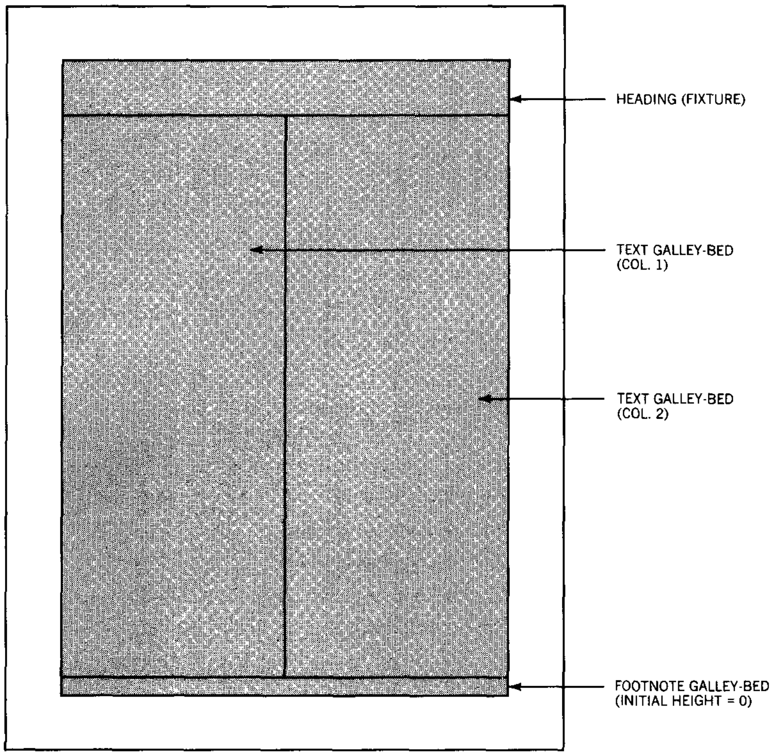
The next step in formatting the document is to actually place the slugs from the galleys onto pages. This process is performed by a "packer" program under the control of "page templates" provided as part of the document profile. An example of a page template is shown in Figure 6. In a page template, the rectangular area of the page is divided up into a set of named "galley-beds" and "fixture-beds." In addition to their positions on the page, these "beds" have properties that control how their boundaries can move as the page is packed with slugs. For example, Figure 6 shows a FOOTNOTE galley-bed which is initially of zero height but can grow upward as it fills with slugs from the FOOTNOTE galley; as it grows upward, the TEXT galley-bed shrinks to make room. Slugs are packed on the page in the order they are emitted by the tag routines. If, for example, a page-wide TEXT slug is encountered, it is laid across both columns of the TEXT galley-bed (such a slug would probably be accompanied by a command to balance the columns above the wide slug). In order to accommodate the packing of slugs, all the columns of a galley-bed must have the same width (but different galley-beds may have different column widths).

When the next slug to be packed will not fit on the current page, a new copy of the page template is invoked. First, the fixture-beds on the new page are packed with the latest "fixture" slugs emitted by the tag routines; next, any "floating" slugs that may have floated forward from earlier pages are packed; then, packing of slugs from the galleys continues in normal order.

A given document may have several page templates—e.g., one for a title page, one for the body of the document, and one for appendices. Switching from one template to another is controlled by commands in the tag routines.

In the JANUS system, page templates are created by an interactive graphic interface. The document designer describes the desired

Figure 6 A page template

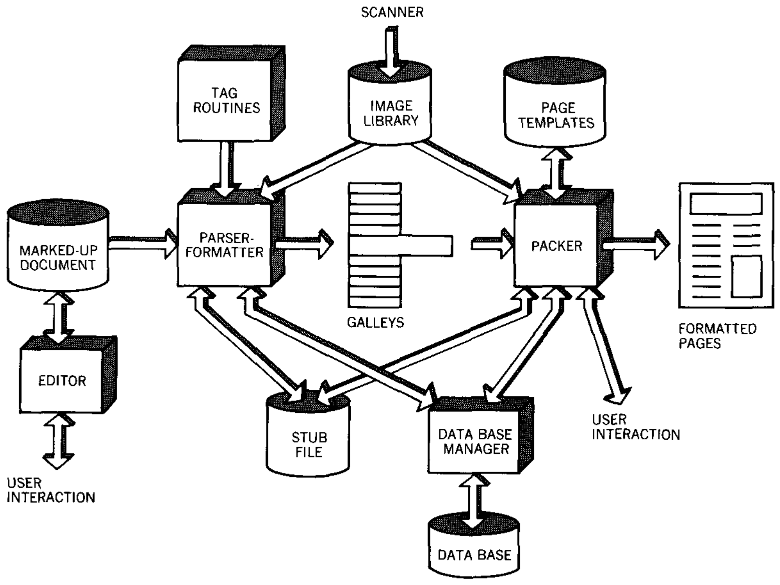


template by filling in a menu on his IBM 3277 display; simultaneously, the system draws a picture of the template on the attached graphic display for verification.

The JANUS architecture

During a JANUS session, the user edits his document source file using a conventional editor on an IBM 3277 display terminal, while formatted pages of the document are displayed on an attached storage-tube display (the IBM 3277 Graphics Attachment). JANUS does not reformat the document after every editing command, because this would cause unnecessary delays and because the document may pass through various inconsistent states as the user makes a series of related changes. Instead, JANUS implements a "SHOW" command whereby the user may request to be shown either (a) the page containing the current line of the input file or (b) any specific numbered page. Thus, rather than proceeding through the document in a linear fashion from front to back, the JANUS user may choose to skip around in the document, applying changes in any desired order and observing their effects.

Figure 7 JANUS architecture



In order to permit the user to skip from one place to another in a document, JANUS must be able to restart the formatting process in the middle of an input file. This procedure is made possible by periodically saving the system's internal state in a file called a "stub," which can later be reloaded if the user's attention returns to this part of the document. This process is made more complex by the two-phase nature of the JANUS system, as illustrated in Figure 7. The definer of a new document type provides JANUS with a set of tag routines and a set of page templates. The JANUS parser/formatter accepts an input document, recognizes tags, and processes them using the given tag routines, producing one or more "galleys." The galleys are then electronically cut into pieces and packed onto page templates by the JANUS "packer" program.

The parser/formatter (hereafter called simply the "formatter") periodically saves its state in an "F-stub," and the packer periodically saves its state in a "P-stub." However, since the formatter and the packer are asynchronous coroutines, the F-stubs and the P-stubs are not perfectly correlated. The relationship between the two kinds of stubs is recorded in the following way:

1. Whenever the formatter makes an F-stub, it "marks" the input file (placing an invisible label on the current line of text) and attaches the name of the F-stub to the current slug in the galley.
2. The packer is aware of the locations of F-stubs by reading the galley. The packer makes a P-stub at the beginning of every page.

For each P-stub, the packer records the name of the nearest preceding F-stub and the number of slugs in the galley between the F-stub and the P-stub.

The JANUS system can then be restarted on an arbitrary page by the following process:

1. Load the P-stub for the desired page into the internal state of the packer.
2. Find the name of the nearest preceding F-stub and reload it into the internal state of the formatter.
3. Reposition the editor to the "mark" associated with the chosen F-stub. (This is more complex than it sounds because the marked line may have been moved or deleted. In this case, it may be necessary to use an earlier P-stub and F-stub.) It is necessary to position the editor to the point where the F-stub was taken because the editor is used to materialize input lines for formatting.
4. Restart the formatter and packer, instructing the packer to disregard the number of slugs between the F-stub and the P-stub before beginning to pack the desired page.

Degrees of safety in interactive formatting

In our study of interactive formatting, we quickly observed that a very small change to a source document can cause a great deal of formatting work to be done. This effect is mainly due to the complex interdependencies within a document caused by forward and backward references. For example, insertion of a single character may cause material to spill onto the next page, which in turn ripples forward, changing many page boundaries and affecting the table of contents and references throughout the document. Two or more passes through the entire document may be required before all page references are resolved to stable values. Clearly, it is not feasible to do this much work for every SHOW command in an interactive system. Therefore, JANUS allows the user to select one of three "degrees of safety" on each SHOW command, with the following definitions:

- *FAST:* Data on the source and format screens will locally correspond. However, there may be errors in pagination or in backward references due to changes occurring earlier in the source that have not been reformatted.
- *SAFE:* Everything is correct except for forward references (and possible side effects caused by incorrect forward references). SAFE may be thought of as "best single-pass formatting."
- *PERFECT:* Everything is formatted correctly, and all references are resolved correctly. In general, PERFECT formatting requires multiple passes.

Implementation of the "FAST" degree of safety is relatively straightforward. The JANUS system is simply restarted at the nearest (P-stub, F-stub) pair available before the desired page and allowed to proceed forward until the desired page has been formatted, packed, and displayed.

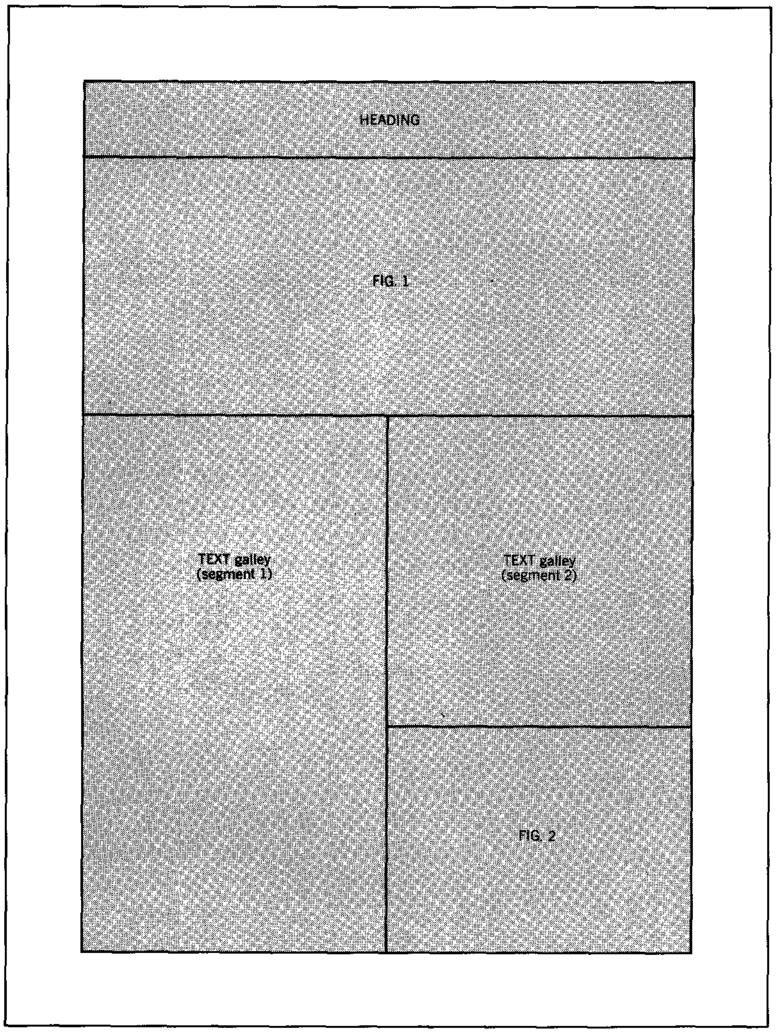
Efficient implementation of the "SAFE" degree of safety is somewhat more complex. Clearly, the command can be implemented by simply starting over again from the beginning of the document. However, there may be several (F-stub, P-stub) pairs available that allow us to start formatting at various internal points in the document. It is not sufficient to find a stub pair close to but prior to the desired page; we must find one that is both prior to the desired page *and* that lies in that portion of the document which is currently "SAFE"—i.e., that portion of the document lying before the earliest change made since the last "SHOW SAFE" command. The state of formatting described by the stub pair at that point is then guaranteed to be the same as what would be derived by starting formatting again at the beginning of the document.

A JANUS service routine examines the "change flags" that are maintained by the editor, and finds the "First Change Point"—the first line number where an editing change has occurred since the last "SHOW SAFE" command. The system also maintains a "Safe Mark," which is defined to be the highest-numbered page that is currently formatted SAFE. The "SHOW SAFE" command is implemented as follows:

1. If the First Change Point is earlier than the Safe Mark, the Safe Mark is moved back to the page before the First Change Point.
2. If the desired page is earlier than the Safe Mark, it can be displayed immediately without reformatting. Otherwise, JANUS is restarted from a (P-stub, F-stub) pair at the Safe Mark and allowed to run forward until the desired page is reached and displayed.
3. If the displayed page is beyond the Safe Mark, the Safe Mark is advanced to the displayed page.

Implementation of the "PERFECT" degree of safety, as noted above, may in general require multiple passes through the document to ensure that all page references have converged to a stable state. To detect this state of convergence, JANUS uses a data base manager with certain special features. The location of each figure, chapter heading, or other object that can be the target of a reference is stored in the data base as soon as the object is encountered by the packer. Tags that need to refer to the object can then find its page number by looking in the data base. Each entry in the data base has three special flags, maintained by the data base manager, called the "read," "write," and "change" flags, which are set respectively when the entry is read, written, and updated to a different value. (Note that it is possible to rewrite an existing entry without changing it.) When a data item is read and subsequently changed, the data base manager

Figure 8 Layout mode display



turns on a "warning" flag, which denotes the fact that a data value used in formatting has become invalid. To achieve the "PERFECT" degree of safety, JANUS proceeds to the end of the document using the rules for SAFE formatting. It then inspects the "warning" flag. If the warning flag is off, the formatting is "PERFECT." Otherwise, JANUS resets all data base flags, makes another complete pass through the document, and examines the warning flag again. Although it is possible to construct pathological cases that never converge, it is expected that "PERFECT" formatting will be achieved after no more than two passes in most cases.

Interactive page layout

Ordinarily, we expect that the information contained in the tag-definitions and the page-templates will enable the formatter to do an acceptable job of page layout. However, for very complex documents, the user may occasionally wish to overrule the system's decisions and take direct control of the format of a page—e.g., specifying a particular placement for a figure. JANUS will permit users this degree of direct control by means of a feature called "layout mode." In layout mode, the output display shows a graphic representation of the galley-beds on the current page, together with the current position of all "floating" objects (e.g., figures). An example of a layout mode display is shown in Figure 8. The user may revise the layout of the page by pointing to various objects, using a joystick-controlled cursor, and issuing commands. For example, a floating figure may be moved from one position to another or deleted from the page, areas of white space may be specified, or two figures may be bound to designated positions on the same page. The page layout created by the user in layout mode serves as input to the JANUS packer, which repacks the slugs on the page according to the new specifications, causing text to flow around the objects placed by the user. The user-specified page layout is saved in the form of a "special template," similar to the regular page-templates in the document profile, keyed by the names of the floating objects on the page. When the packer encounters one of these floating objects later in the session or in a subsequent session, it automatically reinvokes the corresponding special template. A command is also provided to cancel a special template and revert to default layout for the indicated page.

The user may toggle back and forth between layout mode and interactive editing mode as many times as necessary until the page is satisfactory. Of course, changes made to the format of a given page may affect the format of subsequent pages, so the user is advised to proceed through the document from front to back when making final adjustments to page layouts.

Summary

We have discussed the architecture of a document composition system that offers the following principal features:

- It is highly interactive, providing authors with immediate feedback by means of an all-points-addressable display.
- It is capable of formatting complex documents containing mixtures of text, images, and graphics.
- It allows users to mark up their documents with high-level descriptive tags.
- It provides a powerful and easy-to-use interface for defining the meanings of tags and specifying how various objects are to be placed on pages.
- It provides a two-screen workstation in which the user can see both his original markup and the resulting formatted pages simultaneously.

- It provides a set of graphic commands by means of which the user can take direct control over page layout when necessary.

An experimental prototype based on the JANUS architecture is currently under construction at the IBM Research Laboratory in San Jose. In addition to demonstrating the function discussed in this paper, the prototype will be used as a base from which to explore related issues such as formatting tables and equations and interaction with a data base system that will permit the imbedding of material such as bibliographic references and computer-generated data.

ACKNOWLEDGMENTS

The authors wish to acknowledge the contributions of John Bakopoulos and Richard Furuta to the design and implementation of the JANUS prototype.

Portions of this paper appeared in *Proceedings of the ACM SIGPLAN/SIGOA Symposium on Text Manipulation* (Portland, Oregon, June 1981; ACM Order No. 548810) and are reprinted with permission of ACM.

Appendix: Sample tags

The sample tag routines shown here implement two simple tags, LIST and ITEM, for formatting numbered lists. In the case of a list inside another list, these tag routines might produce the proto-galley shown in Figure 4.

```
procedure list;   (* tag routine for numbered list *)

   begin

      listlevel := listlevel + 1;
      (* Listlevel is a global variable which counts nesting level
         of lists.  It is initialized to zero by the outermost
         document tag. *)

      if listlevel > maxlevel then
      begin

         tagerr ('Maximum list level exceeded.');
         (* Imbeds an error message in the document
            and skips over the scope of the tag. *)

      end
      else
      begin

         listcount[listlevel] := 0;
         (* listcount is a global array holding the current
            item count at each list level. *)

         janparse;
         (* Directs JANUS to continue parsing the document, calling
            other tag routines as tags are encountered, until
            the end of the scope of the current (LIST) tag. *)
      end;

      listlevel := listlevel-1;

   end;   (* LIST tag routine *)

procedure item;   (* tag routine for item in numbered list *)

   begin
      if listlevel > 0 then
      begin

         box ('NAME=ITEM TOPHINGE=1L BOTTOMHINGE=1L'
              || ' TOPSHIELD=2L BOTTOMSHIELD=2L');
         (* Creates a box to hold the list item.  Specifies one-line
            hinges to separate the item from adjoining text, and
            two-line shields to prevent "widow" lines. *)

         box ('NAME=MARKER LEFT=LEFT(ITEM) WIDTH=5M');
         (* Creates a box to hold the number of the list item,
            left-aligned and of width 5 "ems" in the current font. *)

         listcount[listlevel] := listcount[listlevel] + 1;

         case listlevel of
            1: format(numeric(listcount[listlevel]));
            2: format(alphabetic(listcount[listlevel]));
            otherwise format(roman(listcount[listlevel]));
         end;
         (* NUMERIC, ALPHABETIC, and ROMAN are JANUS-provided
            functions which return list counters in various styles.
            The FORMAT procedure places the counter in the
            current box--i.e., the MARKER box. *)

         endbox;   (* Closes the MARKER box. *)

         box ('NAME = ITEMBODY LEFT=RIGHT(MARKER) TOP=TOP(ITEM)');
         (* Creates an indented box called ITEMBODY to hold the
            actual text of the item as well as any nested tags. *)

         janparse;
         (* Pours the scope of the ITEM tag into the current box,
            calling other tag routines as tags are encountered. *)

         endbox;   (* Closes the ITEMBODY box. *)

         endbox;   (* Closes the ITEM box. *)

      end
      else   (* listlevel <= 0 *)
      begin

         tagerr ('List item is not inside a list.');
         (* Imbeds an error message in the document and
            skips over the scope of the tag. *)

      end;
   end;   (* ITEM tag routine *)
```

CITED REFERENCES AND NOTE

1. *Document Composition Facility User's Guide,* SH20-9161, IBM Corporation (April 1980); available through IBM branch offices.
2. D. E. Knuth, *TEX: A System for Technical Text,* American Mathematical Society, Providence, RI (1979).
3. B. K. Reid, "A high-level approach to computer document formatting," *Conference Record of the Seventh Annual ACM Symposium on Principles of Programming Languages,* Las Vegas, NV (January 1980), pp. 24–31.

4. *IBM Displaywriter System Operator Reference Guide,* S544-0859, IBM Corporation (March 1981); available through IBM branch offices.
5. "Xerox's Star," *The Seybold Report* **10,** No. 16, 1 (April 1981).
6. "New breakthroughs in halftone generation," *The Seybold Report* **9,** No. 24, 3 (August 1980).
7. "Electronic manipulation of color imagery," *The Seybold Report* **10,** No. 17, 3 (May 1981).
8. *Document Composition Facility, Generalized Markup Language,* SH20-9188, IBM Corporation (April 1980); available through IBM branch offices.
9. *IBM 3277 Display Station, Graphics Attachment RPQ 7H0284, Custom Feature Description,* GA33-3039, IBM Corporation (July 1979); available through IBM branch offices.
10. *PANEL2 Users' Guide,* SH20-2521, IBM Corporation (March 1981); available through IBM branch offices.
11. K. Jensen and N. Wirth, *Pascal: User Manual and Report,* Springer-Verlag, New York (1974).
12. *Pascal/VS Language Reference Manual,* SH20-6168, IBM Corporation (April 1981); available through IBM branch offices.
13. Readers familiar with GML will notice that the tag routines in the Appendix do not have the same names or behavior as the GML "starter set" tags for ordered lists. The sample tag routines have been simplified and heavily commented for clarity of reading. The JANUS system could be used to define tags that closely imitate the GML starter set, as well as to define other tags for various purposes.

D. D. Chamberlin, O. P. Bertrand, M. J. Goodfellow, J. C. King, and B. W. Wade are located at the IBM Research laboratory, 5600 Cottle Road, San Jose, CA 95193, D. R. Slutz is at ESVEL, Inc., 750A Camden Avenue, Campbell, CA 95008; S. J. P. Todd is at the IBM United Kingdom Scientific Centre, Athelstan House, St. Clement Street, Winchester, Hants SO23 9DR, England.

Reprint Order No. G321-5169.