ID Workbench IBMIDDoc User’s Guide and Reference Release 3.6

SH21-0783-10

October 30, 2001

Mike Temple

ID Workbench

IBMIDDoc User's Guide and Reference

Release 3.6

SH21-0783-10

ID Workbench

IBMIDDoc User's Guide and Reference

Release 3.6

SH21-0783-10

Note

Before using this information, be sure to read the general information under Appendix C, "Notices" on page 467.

This manual was produced using IBMIDDoc SGML, the Epic editor, and processed for print and online using the ID Workbench.

Tenth Edition, October 30, 2001

This edition applies to the IBMIDDoc language, version 4.3.6; and to ID Workbench, release 3.6, and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About This Book

This book describes how to use IBMIDDoc, which is a document markup language based on Standard Generalized Markup Language (SGML).

Who Should Read This Book

Anyone who wants to create documents with IBMIDDoc markup or design a library of documents that have IBMIDDoc markup should read this book.

If you are new to SGML and to the ID Workbench, please first get and read the *IDWB Getting Started and User's Guide*. Access the latest versions of these books from the ID Workbench Documents page:

`http://w3.rchland.ibm.com/projects/IDWB/documents/idwbdocs.htm`

What You Should Already Know

You should be familiar with the process of creating a document and the general concepts of document markup. You should also know how to use an SGML editor.

Although IBMIDDoc markup can be entered using a text editor, an SGML editor such as the Arbortext Epiceditor or Frame2000 is strongly recommended. The SGML-aware editors ensure your markup is correct before you do any formatting; thus saving you time and money in extra formatting and debugging runs.

How This Book is Organized

This book is organized into the following parts:

- Part 1, "Introduction to IBMIDDoc" on page 1 describes basic IBMIDDoc terms and concepts and IBMIDDoc markup rules.
- Part 2, "Using IBMIDDoc Markup" on page 15 describes how to use IBMIDDoc markup to create the different parts of a document.
- Part 3, "IBMIDDoc Markup Reference" on page 221 describes the markup for IBMIDDoc elements and attributes.

If you are planning or designing libraries or information, you should be generally familiar with IBMIDDoc and be very familiar with the following information, which is essential to planning and designing information:

- Chapter 1, "Introduction to IBMIDDoc" on page 3
- Chapter 2, "Using basic IBMIDDoc elements to create a document" on page 19
- Chapter 12, "All about linking" on page 129
- Chapter 16, "Developing Programming Language Reference Materials" on page 165
- Chapter 18, "File, text, and character entities and reusing information" on page 179
- Chapter 19, "Conditionally including information" on page 195

Summary of Changes

Changes for IBMIDDoc version 4.3.6 and IDWB release 3.6:

- You have more flexibility on setting page, column, or text wide items; see the PGWIDE attribute on "Fig (figure)" on page 281, "Screen (display screen)" on page 419, "Syntax (syntax diagram)" on page 428, "Xmp (example)" on page 449, and "CGraphic (character graphic)" on page 247.
- You can more easily create architected information, using the metadata tag; see "Creating an information architecture" on page 27.
- The XHTML output transform includes a way of passing classes of elements through to style sheets. This allows a setting in the document to be reflected as a style sheet setting. See "Using document classes with XHTML style sheets" on page 51.
- You have an improved way of specifying simple lists, see "Simple lists" on page 30.
- You can better control the highlighting of terms and headings on definition and parameter lists; see "Definition lists" on page 32 and "Parameter lists" on page 34.
- You can control the size of list item dingbats; see "Scaling list dingbats" on page 36.
- There's a new explanation of how to create multi-part figures; see "Multipart figures" on page 58.
- You can control the shading for tables, table rows, or table cells; see "Affecting how a table appears: Rules, Separators, Shading" on page 72.
- There's a new explanation of how to make your XHTML tables accessible to screen readers; see "Making your tables accessible" on page 76.
- You can override or change the number of columns (the layout) of your document, chapters, or chapter-like divisions; see "Getting in style, the document style, that is" on page 83.
- You can use the new definition tags to create defaults for corresponding tags; see Chapter 9, "Using definition tags" on page 105.
- You can set more levels of active revisions in Xyvision PostScript or PDF documents.
- The "See" and "See also" index entries now work in Xyvision PostScript or PDF documents; see "Defining See and See-also references" on page 121.
- There is a new description of index sorting; see "Controlling the Index Sorting" on page 123.
- You can have index entries appear as part of an online review; see "Helping online reviewers see your index entries" on page 124.
- There is an improved explanation of cross-book linking; see "Linking to items in another IBMIDDoc document" on page 133.
- You can scale text larger or smaller; see Chapter 21, "Making some things bigger or smaller" on page 205.
- You can control the starting page number and chapter number for divisions; see "D (hierarchical division)" on page 261.
- You can now show syntax diagrams without intervening spaces; the "composite" value was added; see "The Group element" on page 150.

| • A list of IBMIDDoc input codepages is now included; see "IBMIDDoc Input
| Codepages" on page 12.

Changes for IBMIDDoc version 4.3.5 and IDWB release 3.4:

- Updates after October 12th:
  - New symbols for the e(logo)server logos; see "Special characters" on page 180.
  - New BRAND and NEWBRAND attributes added to the IBMIDDoc tag; see "IBMIDDoc (IBM-specific product documentation)" on page 299.
- Add MMOBJ to the content model for Screen. Requirement R004878.
- Line justification for DBCS languages: ibmiddoc style="xpp:(justify)". This is used only for DBCS languages. Requirement R005448.
- RETKEY=None | First | Last | FirstLast | NoDup on LERS, MSGLIST, and GL. Used to enable or disable the automatic running heads for the LERS, Msglist, GL in a document. Cannot set at the individual element level. All explicitly coded Retkey elements are honored. If you nest elements that can generate a running head (for example: msglist in lers), only the outer active generated head is used. That is, if you have specified automated retkey generation for LERS and MSGLIST, then a MsgNo inside LERs would not be used in the retkey area. But if you had an explicit retkey inside the msg, then the retkey is honored as an explicit override. The Xyvision transform will only use one style of retrieval per retkey type First for Lers; NoDup for MsgList and GL. Requirement RALMAR04.1997a.
- IBMIDDoc MAXTOC=number to enable you to specify the highest level head to go in the Table of Contents. MAXTOC was picked to distinguish it slightly from the Bookie :docprof toc=123 which actually let authors skip some heads altogether but still get lower level heads. Requirement R004769.
- D toc=toc|notoc also for special D type elements. Controls whether this particular heading is included in the TOC if the TOC includes those levels of headings.
- Change PGWIDE values for TABLE: pgwide=0|1|2. The new value 2 on the pgwide is to indicate width=textline behavior.
- Add FRAME attribute to Fig Frame: Fig frame=NONE|BOX|RULES. Requirement R004763.
- Add SynStyle to syntax: SynStyle= Space| Box| Rule| LblBox. Requirement R004763
- Add one and two character termwidth settings to DL and PARML. The defined attribute values for termwidth are now small | medium | large | 1 | 2. Requirement R005298.
- Change processing of DVCFOBJ to error message.
- Add DBLK to support including multiple divisions from an object library. It is allowed wherever D is allowed.
- Add PBLK to content model for FrontCover, to allow for multiple paragraphs and label boxes on cover. Requirement R005266
- Add LAYOUT to TOC, FIGLIST, TLIST, and Index. This is to support using less columns than defined in the default style to provide room for long terms. The values will be onecol, twocol, threecol, and default-layout. Requirements R005530 and R005509
- Provide support for index folio prefix used for multiple volumes: Added MULTIVOL=OneVol | Index-Folio to IBMIDDOC , which will add X- as a prefix for the page numbers in the index and start the numbering from 1. Requirement R004964.

- Added comments to DTDs with Language and DocStyle values currently supported.
- Allow compact lists: LINESPACE=SPACE|COMPACT on all lists (ol ul gl msglist codel parml dl notel), sublists automatically inherit the linespace but can override. We chose this attribute rather than COMPACT=COMPACT|NOCOMPACT to allow for future growth like doublespace. Requirement R005298
- With patch IDWXF036, the CONLOC attribute passes the attributes from the elements contained in an OBJLIB. See “Reusing elements from an object library” on page 191 for more information.

Part 1. Introduction to IBMIDDoc

Chapter 1. Introduction to IBMIDDoc

This chapter introduces IBMIDDoc and describes IBMIDDoc terms and concepts. It also includes considerations and rules for IBMIDDoc markup.

What is IBMIDDoc?

IBMIDDoc is a document markup language based on Standard Generalized Markup Language (SGML). SGML is an international standard for representing the elements and structure of electronically stored information so that a person or computer program can understand and use those elements and structure. The electronically stored information can be one or more files that make up a document.

IBMIDDoc Documents

An IBMIDDoc document is a valid SGML document. A valid SGML document is comprised of:

- A document type declaration that contains or references a document type definition (DTD)
- A document instance (your text) which conforms to the DTD contained in or referenced in the document type declaration

The IBMIDDoc DTD must be referenced by all IBMIDDoc documents. Figure 1 illustrates a valid IBMIDDoc document type reference.

```
<!DOCTYPE IBMIDDOC PUBLIC "+//ISBN 0-933186::IBM//DTD IBMIDDoc//EN" >
```

Figure 1. Document Type Declaration for an IBMIDDoc Document

Figure 1 shows the document type declaration, which names the document type (IBMIDDoc). It also references the PUBLIC identifier for the DTD. The public identifier is a name that uses a format defined by the SGML standard. This name format allows us to point to information in a system-independent way. The SGML application that is processing the SGML data uses the identifier to transform the data being read to an identifier that works on the SGML system being used.

The document instance must conform to the document type definition. For IBMIDDoc documents, this is the IBMIDDoc DTD. Figure 2 shows the absolute minimum IBMIDDoc document.

```
<!DOCTYPE IBMIDDoc public "+//ISBN 0-933186::IBM//DTD IBMIDDoc//EN">
<IBMIDDOC>
<body>
```

Figure 2. Minimum IBMIDDoc Document Markup

Figure 3 on page 4 shows a more complete, but still small, IBMIDDoc document:

```
<!DOCTYPE IBMIDDoc public "+//ISBN 0-933186::IBM//DTD IBMIDDoc//EN">
<ibmiddoc>
<body>
<d>
<dprolog><titleblk>
<title>My Little Document</title>
</titleblk></dprolog>
<dbody>
<p>This is my first sample document. Thank-you for
reading it.</p>
</dbody></d>
</body>
</ibmiddoc>
```

Figure 3. IBMIDDoc Document

| When you save your document, give it a meaningful file name. The file extension
| needs to be IDD (or idd), to ensure the ID Workbench processes properly recognize
| the file. Use only letters and numbers in the file name; we recommend starting the
| file name with a letter. Do not include special characters in the file name (such as
| spaces, +, –, %, and so forth).

IBMIDDoc Terms and Concepts

This section introduces IBMIDDoc terms and concepts, including markup and tags, containment, entities, object libraries, attributes, property and class definition, and separation of content and style.

Documents and Markup

A ***document*** is a collection of information that is processed as a unit. An IBMIDDoc document consists of information (text and graphics) and IBMIDDoc markup that defines and identifies the structure and the elements of the document.

The IBMIDDoc ***document type definition*** (DTD, not "DDT") defines the document type and the valid elements or tags you can use. Your document that contains the information (text and graphics) and corresponding markup is called the ***document instance***.

Markup is information in a source document that enables a person or system to process the document. The kinds of markup you can use in IBMIDDoc are: descriptive markup (tags), markup declarations, entity references, marked sections, and processing instructions.

Elements and Tags

An ***element*** is a component of a document such as a paragraph, an unordered list, or a figure. ***Tags*** (or ***descriptive markup***) are used to identify elements. A tag is composed of a tag open delimiter (< for a start tag), an element identifier (for example, p for a paragraph), and a tag close delimiter (>). Ending tags begin with </, an element identifier, and >.

In the following example of a paragraph, <P> marks the start of the paragraph element and </P> marks the end of the paragraph element:

```
<P>This is the content
of a paragraph element.
</P>
```

The ***content*** of an element is whatever is between the start and end tags for the element. An element can contain information (text or multimedia objects), other elements, or a mixture of information and elements. Elements that have no content do not have end tags, such as the XRef element.

Containment

One element can contain another element either directly or indirectly, known as ***direct containment*** or ***indirect containment***, respectively.

In the following example, the first paragraph directly contains an ordered list, the ordered list directly contains two list items, and the first list item directly contains a paragraph. The first paragraph indirectly contains the list items and the paragraph contained by the first list item.

```
<p>This is a list:
<ol>
<li>First list item
<p>This is a paragraph within the
first list item.</p>
</li>
<li>Second list item</li>
</ol>
</p>
```

This next example, the first paragraph does not contain the ordered list, because the first paragraph is closed before the list is opened.

```
<p>This is a list:
</p>
<ol>
<li>First list item
<p>This is a paragraph within the
first list item.</p>
</li>
<li>Second list item</li>
</ol>
```

The containment structure also determines the inheritance of properties from an element to the elements it directly contains (its children). This structure also determines the properties of elements that are indirectly contained by other elements.

Containment also determines a hierarchical structure. Divisions within divisions determine headings and subordinate headings. Several SGML editors can display a tree view of a document. This view lets you see these containment and hierarchical relationships. You can tell which elements are peers, parents, or children by this kind of view.

Entities

An ***entity*** is any information that is referred to as a unit from a document. It can be a character string, a file, a graphic, or a collection of files. It can even be an entire document. Entities enable the reuse of information, and organization of that information into separate files.

An entity must be defined by a ***markup declaration***, which is a kind of markup that controls the interpretation of other markup. Entities are not dynamic; the definition of an entity cannot be changed after it is defined. In addition, all entities must be defined at the beginning of a document. Entity declarations are part of the DTD. The declarations are usually put in the DTD subset, which is the part of the DTD that is specific to a given document.

An ***entity reference*** requests that entity data replace the entity reference at the place where the reference occurs. Entity references are delimited by the entity reference open delimiter (&) and the entity reference close delimiter (;).

Entities are either internal or external. An ***internal entity*** is an entity whose declaration includes the replacement text (the text that is to replace the entity reference). In the following example, which shows how to define and refer to an internal entity, `IBMIDDoc` is the replacement text and `&product;` the entity reference.

```
<!ENTITY product "IBMIDDoc">
⋮
This book teaches you how to use &product;.
```

Internal entities that are defined by IBMIDDoc include special characters such as the em-dash or backslash, which are referred to with the entity references `&emdash;` and `&bslash;`, respectively.

These entities are contained in the `IDDBKSYM.ENT` file. Many of these character entity declarations use the same names as the character entities that are defined by BookMaster®.

An ***external entity*** is an entity whose declaration defines where the replacement text can be found but does not include the replacement text in the declaration. These are more commonly called symbols (replacement words or phrases) or imbeds (files such as a chapter).

In the following example, which shows how to define and refer to an external entity, the XXXL0INT.IDE file is the external entity and `&introfile;` is the entity reference:

```
<!ENTITY introfile SYSTEM "xxxl0int.ide">
   ⋮
&introfile;
```

The ENTITY line defines ″introfile″ as the name of the entity; ″system″ indicates the following string ″xxxl0int.ide″ is a file name.

The example that follows shows the entity declarations for several IBMIDDoc files that are included in a master file document. This is like using the `.im` Bookmaster macro to imbed SCRIPT files in a master document.

```
   ⋮
<!Entity  RKTL1EDN   SYSTEM 'RKTL1EDN.IDE'>
<!Entity  RKTL1NOT   SYSTEM 'RKTL1NOT.IDE'>
<!Entity  RKTL1PRE   SYSTEM 'RKTL1PRE.IDE'>
   ⋮
<!-- Edition notice This includes the entity containing Edition Info-->
&RKTL1EDN;
<TOC><GENDTITLE>
<!--  Notices This includes the entity containing the Notices -->
&RKTL1NOT;
<!--  Preface This includes the entity containing the EdNotices -->
&RKTL1PRE;
</FRONTM>
```

In the first line of the example, `RKTL1EDN` is the entity name, `SYSTEM` declares where the entity is stored on the local system, and `'RKTL1EDN.IDE'` is the actual file name. In the front-matter portion of the document, the entities are referenced where they should appear during processing.

For more information about using entities and entity references, see Chapter 18, “File, text, and character entities and reusing information” on page 179.

Epic Editor Note

For information about creating, declaring, and referencing internal and external entities using Epic, see the chapter “Editing SGML Documents with Epic” in the *ID Workbench Getting Started and User’s Guide*. If you’re not using Epic, see the user guide for the SGML editor that you are using.

Migration Note

Internal entities are ″symbols″ in BookMaster*, and external entities are ″embedded files″ in BookMaster.

You will often see entities referred to in the following ways:

Character Entity
Contains values for a special character set, as in the IDDBKSYM.ENT file.

Text Entity
Contains the replacement text in the markup declaration.

File Entity
Contains a reference to the name of another file that you want to reference within your document.

Marked sections

Marked sections are a special way of controlling a part of a document for processing. You can indicate, in the source, a part to include or ignore. IBMIDDoc has a better way of doing this at run time; using properties; see Chapter 19, “Conditionally including information” on page 195.

To use a marked section to condition text (maybe to hide text and SGML source you want to save but not have in the document), You use a marked section parameter to surround the text. For example, this is hiding a paragraph; notice the brackets and the %comment – these cause the tagged content to be ignored.

```
<![ %comment; [<p>Here's a little paragraph I want to hide.</p>]]>
```

The %comment needs to be declared,and set to ignore:

```
<!ENTITY % comment "IGNORE">
```

Processing instructions

These are special instructions that you add to your source. They are allowed almost anywhere. You use them within IBMIDDoc documents to include special formatting controls such as page breaks.

For example, the following processing instruction tells the Xyvision formatter to start a new page:

```
<?IDD:page>
```

You can force new pages, which should be used as little as possible. Place this processing instruction outside of elements if possible. The syntax is:

```
<?IDD:page>
<?IDD:page odd>
<?IDD:page even>
<?IDD:page x.x>
```

where "page" forces a page eject; "page odd" forces a page eject to an odd-numbered page; and "page even" forces a page eject to an even-numbered page. "page x.x" specifies a measurement up from the bottom of the current page (above the running foot). Supported measurements are: in (inches), pi (picas), and cm (centimeters). Just specifying a number indicates the number of lines in the current font. For example, this indicates that a new page should start if less than 4.5 centimeters remain in the current page:

```
<?IDD:page 4.5cm>
```

This specifies 3 picas and 6 points:

```
<?IDD:page 3.6pi>
```

You can also force new lines, which should be used as little as possible. The syntax is:

```
<?IDD:break>
```

For example:

```
<p>Here is some text
<?IDD:break>that should start on a new line.
```

Object Libraries

An ***object library*** is a collection of elements that can be used elsewhere in a document. Object libraries, like entities, also enable reuse of information. Elements in an object library can be used only within the document that contains the object library. Object libraries can also be used for conditional processing. Conditional processing allows you to turn text on or off as you process your document.

For more information about using the elements in an object library, see Chapter 18, "File, text, and character entities and reusing information" on page 179 and Chapter 19, "Conditionally including information" on page 195.

Migration Note

Object libraries provide the function of BookMaster document version control facility (DVCF) side files.

Attributes

An ***attribute*** is a characteristic of an element (other than type or content) that is included with a start tag to further describe the element. Many attributes defined by IBMIDDoc are common to all elements.

In IBMIDDoc, an attribute name must:

- begin with an alphabetic character, `A-Z` or `a-z`
- contain only `A-Z`, `a-z`, `0-9`, `-` (hyphen), `.` (period), and `_` (underscore)

- be no more than 64 characters in length

IBMIDDoc attributes are divided into the following classes:

- ***Identifying attributes*** identify a given element. The ID attribute, which is a common attribute, is an identifying attribute.

  ```
  <p id="fred">This paragraph has an identifier.</p>
  ```
- ***Property attributes*** define the properties of an element, such as its owner or class, and control which elements are to be processed. Language and Props are property attributes.

  ```
  <p props="v2r3">This paragraph is used only for V2R3.</p>
  ```
- ***Link attributes*** define the link relationships between elements. Linkend and Refid are link attributes for the L and XRef elements, respectively.

  ```
  <P>A <L LINKEND="parahead">paragraph</L> is a chunk of information.
     ⋮
  <P>See <XREF REFID="parahead"> for more information about paragraphs.</P>
     ⋮
  <D ID="parahead">
  ```
- ***Style attributes*** define presentation characteristics of an element. In this next example, the OLTYPE attribute says this list should format as a step list.

  ```
  <ol oltype="step">
  <li>Do this.</li>
  <li>then that.</li>
  </ol>
  ```

Property and Class Definition

Properties such as language, status, or classification can be associated with elements and are defined by using property attributes. Also, elements can inherit properties from other elements.

The PropDef element allows you to define one set of properties that can apply to several elements.

The ClassDef element allows you to define element classes that enable processing functions such as creating a detailed glossary or bibliography, generating precise associative links, or automatically indexing certain kinds of information. Element classes can also be used to control the inheritance of element properties.

In many cases, element classes are defined for an entire collection of documents by someone responsible for designing the information in the collection, such as an information designer or planner. If you are working on information for which element classes have been defined, you need to know the class names, the affected elements, and their intended use.

For more information about defining properties, see Chapter 20, “Property and Class Definitions” on page 201.

Separation of Content and Style

The main intent with IBMIDDoc SGML markup is to separate the content from how it appears. The output styles are determined by style “gurus” so that all our documents look alike.

We need to write clearly and consisely; the formatters take care of how the information appears. We don’t need to worry that a second-level heading is in the proper type face and highlighting.

IBMIDDoc Markup Considerations and Rules

This section describes markup considerations for ending elements and omitting tags. It also lists IBMIDDoc markup rules.

Markup and SGML Editors

All discussions in this book about entering tags and other markup are in the context of using a non-SGML text editor to create IBMIDDoc files. With a text editor, minimizing typing is useful, and IBMIDDoc does what it can to keep typing to a minimum. However, you should create and edit IBMIDDoc files with editors that support SGML, like Epic or Frame+SGML.

With an SGML editor, you make selections from menus rather than typing in tags. Thus, minimizing typing is not an issue with SGML editors.

The markup shown in this book is usually the minimum markup required. However, SGML editors often insert omissible tags for elements. Also, SGML editors often insert an optional attribute name when you enter a value for the attribute. Thus, when you request a view that shows the markup in an SGML editor, you can see tags that you did not select or, if you use a text editor, that you do not need to type.

Ending an Element

In SGML markup, an element is ended either by an end tag or by another element that cannot be directly contained by the first element. Elements that contain nothing have no end tags, and some elements have optional end tags.

The paragraph element has an optional end tag. Because a paragraph cannot directly contain another paragraph, one paragraph automatically ends when another paragraph begins, regardless of whether a paragraph end tag is used.

It is almost never wrong to use an end tag. The exceptions are:

- When an element may never have content. Such elements are called **Empty** elements. XRef is an example of an empty element.
- When attributes which force the element to be empty are specified on the element. All elements have several possible such attributes. CONLOC is an example of such a special attribute. LitData’s OBJ attribute is another example of a special attribute.

  In both of these examples, the attributes are content references. They point to other elements by ID, or to other entities by name. The content of the elements or entities that are the target of the content reference are used at the point where the content reference is made.

Migration Note

Bookmaster’s Artwork tag has a name attribute which behaves the same way as described for these special attribute content references.

To determine whether an end tag is required or optional for a particular element, check the description for that element.

Omitted Tags and Implied Elements

In SGML, some start and end tags can be omitted and the corresponding elements can be implied. The omission makes it easier to create IBMIDDoc documents if you are using a text editor. Remember, however, that you may see error messages about the implied elements. The following division element markup demonstrates how omitted tags work. D contains two main elements, DProlog and DBody. DProlog contains TitleBlk and Title which contain the title of the division, and DBody contains the content of the division. You can type the following markup, in which DProlog, Title, and DBody are all automatically implied:

```
<d>Using IBMIDDoc
<p>IBMIDDoc is IBM Information Development's
implementation of the SGML standard for IBM documentation.
```

If you typed the complete markup, it would look like this:

```
<d>
<dprolog><titleblk>
<title>Using IBMIDDoc</title>
</titleblk></dprolog>
<dbody>
<p>IBMIDDoc is IBM Information Development's implementation
of the SGML standard for IBM documentation.</p>
</dbody></d>
```

Start tags can be omitted only for required elements. Because TitleBlk, Title, DProlog, and DBody are required elements on the D element, you can omit the start and end tags. For each element with an optional title, you must explicitly enter the TitleBlk start and end tags. To determine whether an element is required in a particular context, check the description for the parent element.

Markup Rules

General markup rules for IBMIDDoc are as follows. This first set are good general rules:

- Always use the appropriate markup to identify a document element. For example, do not use a paragraph tag to create a blank line. If you use markup incorrectly, the output might appear satisfactory when processed; however, if the document is processed with a different processing program from the one you are using, the results may be unsatisfactory.
- Use an SGML comment to indicate strange or interesting markup. This not only helps you when you wonder why your did something, it also helps the poor soul that has to take over your document when you get promoted and move to other assignments. You can also include a comment in a declaration; it is delimited by double hyphens (--), as follows:

  ```
  <!ENTITY product "IBMIDDoc" --This comment describes the entity.-->
  <!ENTITY introfile SYSTEM "xxxl0int.ide" --This comment describes
  the entity and is too long to fit on one line.-->
  ⋮
  <!--This is a comment by itself.-->
  ```

  Do not use double hyphens within a comment because they end the comment.
- To “comment out” a section (so it’s hidden but not deleted), use the marked-section keyword IGNORE. See “Marked sections” on page 7 for an example.

These rules are automagically enforced by any SGML editor; you typically do not have to worry about them:

- Specify elements in the right order. Elements that occur only once in a document must be coded in the order shown in the syntax descriptions.
- Define all entities at the beginning of your document.
- For a multiple-word attribute value or for an attribute value that contains blanks or special characters, enclose the value within single or double quotation marks, as shown here:

  ```
  <PH STYLE="bold italic">
  This should be bold and italic
  </PH>
  ```

  If an attribute value contains apostrophes, use double quotation marks, as follows:

  ```
  <P XREFTEXT="operator's tasks">tasks
  ```

  If an attribute value contains one kind of quotation marks (single or double), use the other kind of quotation marks, as follows:

  ```
  style='color="cyan white" bold monospace'

  style="color='cyan white' bold monospace"
  ```

  If a single-word attribute value does not contain special characters, quotation marks can be used but are not required.
- Do not hyphenate words at the end of an input line.
- You can omit the start tag only for a required element.
- You can use empty end tags as a shorthand way of ending the last element started, as shown for the Phrase element in this example:

  ```
  <P>Phyto-daemon is a example of <PH>Generation-X-speak</>.
  <P>This is another paragraph.
  ```

Phrase-Like and Paragraph-Like Elements

In IBMIDDoc, Phrase-Like (%PhLike) is used to refer to all phrase-like elements. Most %PhLike elements are valid anywhere that plain text is valid. There are a few exceptions to this rule where a few elements have very specific content rules.

Paragraph-Like (%PLike) Elements include the IBMIDDoc elements that are directly containable by division elements.

%DivLike Elements include the IBMIDDoc elements that can be contained (in most cases) at the same hierarchical level as a Division element.

Element Groupings in IBMIDDoc

There are several groups of elements that are referred to using generic names as a shorthand technique in this book. These are called parameter entities.

IBMIDDoc Input Codepages

IBMIDDoc documents have expected input codepage for each language. This ensures the ID Workbench transforms do the proper character conversions and sortings. The codepages are listed for each language as follows:

```
ENGLISH       IBM-850
UKENGLISH     IBM-850
DUTCH         IBM-850
GERMAN        IBM-850
ITALIAN       IBM-850
```

```
FRENCH        IBM-850
SPANISH       IBM-850
PORTUGUESE    IBM-850
DANISH        IBM-850
FINNISH       IBM-850
NORWEGIAN     IBM-850
SWEDISH       IBM-850
CFRENCH       IBM-850
BFRENCH       IBM-850
BDUTCH        IBM-850
BPORTUGUESE   IBM-850
CENGLISH      IBM-850
ICELANDIC     IBM-850
SGERMAN       IBM-850
SFRENCH       IBM-850
SITALIAN      IBM-850
KOREAN        IBM-1363
TCHINESE      IBM-950
SCHINESE      IBM-1386
JAPANESE      IBM-943
CATALAN       IBM-850
TURKISH       IBM-857
GREEK         IBM-813
POLISH        IBM-852
CZECH         IBM-852
SLOVAK        IBM-852
HUNGARIAN     IBM-852
CROATIAN      IBM-852
SLOVENIAN     IBM-852
RUSSIAN       IBM-866
ROMANIAN      IBM-852
BULGARIAN     IBM-915
ESTONIAN      IBM-922
LATVIAN       IBM-921
LITHUANIAN    IBM-921
MACEDONIAN    IBM-855
SERBIAN       IBM-855
THAI          IBM-874
ARABIC        IBM-864
HEBREW        IBM-1255
```

Part 2. Using IBMIDDoc Markup

Chapter 2. Using basic IBMIDDoc elements to create a document

This section describes the placement and use of divisions and other division-like elements in your document. The elements discussed in this chapter include the following:

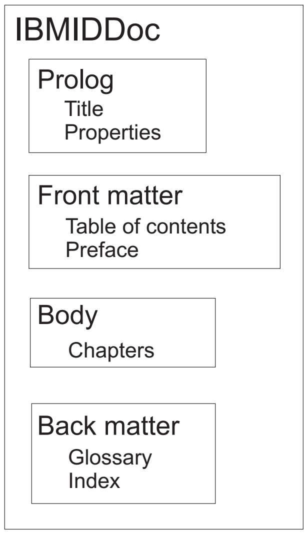
- Body
- D, division
- P, paragraph
- Title
- TitleBlk, title block
- Part

IBMIDDoc Document Structure

The IBMIDDoc DTD defines the rules of structure and containment for all IBMIDDoc elements, and the attributes that can be used on these elements.

At the document level, IBMIDDoc documents can contain the following:

- Prolog element
- FrontM (front matter) element
- Body element
- BackM (back matter) element



Not all of these elements are required in an IBMIDDoc document. When you use an SGML editor, the editor interprets the DTD rules for the correct structure and containment rules for IBMIDDoc, and enforces these rules when you are authoring.

As long as the rules (sometimes called context checking) are active, an SGML editor will only present the IBMIDDoc elements that are valid in the context in which you are editing. An SGML editor will not, for example, allow you to insert a P element directly within another P element.

While there are many aspects to creating an IBMIDDoc document, let's first focus on creating a simple one.

Creating an IBMIDDoc Document

Within the IBMIDDoc element, a IBMIDDoc document must have a Body element, which must contain a division or division-like element. We'll look at these basic elements now, and look at other IBMIDDoc elements in the chapters that follow.

Creating the body of your document

The Body element contains the body of the document. This is where you put the chapters for your document. The body can contain any number of D, LERS, MSGList, Proc, and Part elements. In the example that follows, the Body element contains two division (D) elements. Because the divisions are all contained at the same level, each is a chapter (so this is a simple document with two chapters).

```
<ibmiddoc>
<body>
<d>
<dprolog><titleblk>
<title>My Little Chapter</title>
</titleblk></dprolog>
<dbody>
<p>Here's the beginning of my chapter.</p>
</dbody></d>
<d>
<dprolog><titleblk>
<title>Another little chapter</title>
</titleblk></dprolog>
<dbody>
<p>It was a dark and stormy night...</p>
</dbody></d>
</body>
</ibmiddoc>
```

Creating divisions (D element)

Most often, you will insert a D element after the Body element in your document. The first division in the document body is the first chapter. This is analogous to an <H1> tag in HTML. When you insert a D element, most SGML editors automatically insert the required sub-elements for the division. In IBMIDDoc, the elements that must be included in a D element are:

DProlog
: The DProlog can contain a number of elements, but the only required elements are TitleBlk and Title, which contain the heading text for that division. Stitle is an optional element that indicates a shorted title. For first-level headings, use this Stitle to shorten the running foot. Subtitle is another optional element that does nothing in a book; it's element is defined, but it is not used.

DBody
: The DBody element contains the text elements that comprise the content of the division; that is, the paragraphs, lists, and your golden prose.

When you create a first-level division in the document hierarchy, the text contained in the Title element is displayed as the chapter title. For example, this shows a sample chapter, first-level division:

```
<ibmiddoc>
<body>
<d>
```

```
<dprolog><titleblk>
<title>My Little Chapter</title>
</titleblk></dprolog>
<dbody>
<p>Here's the beginning of my chapter.</p>
</dbody></d>
</body>
</ibmiddoc>
```

Creating paragraphs (P element)

The element you will use most often is P for Paragraph. The P element contains a paragraph, that is, a block of text representing a single idea. A paragraph can contain other elements such as lists. Paragraphs should contain a single idea, and can contain many other elements. In IBMIDDoc, paragraphs cannot directly contain other paragraphs, but they can contain other elements that contain paragraphs.

Here's a sample of a paragraph, in case you mised the examples shown in previous topics:

```
<ibmiddoc>
<body>
<d>
<dprolog><titleblk>
<title>My Little Chapter</title>
</titleblk></dprolog>
<dbody>
<p>Here's the beginning of my chapter.</p>
</dbody></d>
</body>
</ibmiddoc>
```

When creating paragraphs, keep in mind that each paragraph (like many IBMIDDoc elements) is a container. If you do not wish another element (a list or figure for example) to be contained by the current paragraph, you must enter that element after the end tag for the paragraph.

Deciding which elements to use

There is often more than one permissible way to markup the document content. However, with IBMIDDoc, the intent of the markup is important. For example, you could mark up a list as an unordered list:

- LI elements

  List items contain individual list items.
- LIBlk elements

  List item blocks contain logical groupings of list items.
- Bridge elements

  Bridge elements bridge two concepts.

Here's its markup:

```
<ul>
<li>LI elements
<p>List items contain individual
list items.</p></li>
<li>LIBlk elements
<p>List item blocks contain logical
groupings of list items.</p></li>
<li>Bridge elements
<p>Bridge elements bridge two
concepts.</p></li>
</ul>
```

On the other hand, you could markup up the same information using a definition list:

LI elements
List items contain individual list items.

LIBLK elements
List item blocks contain logical groupings of list items.

BRIDGE elements
BRIDGE elements bridge two concepts.

Here's its markup:

```
<dl>
<dlentry><term>LI elementS</term>
<defn>List items contain individual list items.</defn>
</dlentry>
<dlentry><term>LIBLK elements</term>
<defn>List item blocks contain logical groupings of
list items.</defn>
</dlentry>
<dlentry><term>BRIDGE elements</term>
<defn>BRIDGE elements bridge two concepts.</defn>
</dlentry>
</dl>
```

While either way is acceptable and valid IBMIDDoc markup, consistency in deciding how to mark up your information is important to the successful exploitation of IBMIDDoc markup. You need to mark up information according to its intent. Decide which IBMIDDoc markup best describes the type of information you are containing, and use that markup consistently in your information.

IBMIDDoc allows you to separate the markup from the final presentation. ***You should not*** mark up information so that it will "look good" a certain way in:

- a PostScript or PDF file
- an XHTML or HTML set of files
- an IPF panel

If you are consistent in how you mark up your information, the resulting formatted output, for any target medium, will be treated consistently in that medium. This consistency contributes to our customer's satisfaction with the information.

Creating a heading hierarchy

You create subheadings in your document by creating a heading hierarchy. For example, a division may be a chapter title (1st-level heading), a topic (2nd-level heading), a subtopic (3rd-level heading), and so forth. Nested divisions define this division hierarchy. Nested divisions are divisions that are contained within a division. The level of the headings produced is determined by the nesting. At this point in the book, the previous division is a third-level division.

Here's the markup for two nested divisions; the bold shows the second-level division:

```
<d>
<dprolog><titleblk>
<title>My Little Chapter</title>
</titleblk></dprolog>
<dbody>
<p>Here's the beginning of my chapter.</p>
```

```
<d>
<dprolog><titleblk>
<title>My teeny topic</title>
</titleblk></dprolog>
<dbody>
<p>Here's the beginning of a topic.</p>
</dbody></d>
</dbody></d>
```

If you need more the 6 levels of divisions, an editor might say you have "heading-itis".

Migration Note: All other contained divisions will be treated like subheadings are treated in BookMaster. However, unlike headings in BookMaster, IBMIDDoc divisions are automatically arranged according to their hierarchical position in the markup. Each contained division is handled at a lower heading level, so to speak.

Division prologs

After you enter the title, you can enter a number of optional elements in the division prolog, including:

- Approvers
- Authors
- BibEntryDefs, bibliography entry definitions
- CopyrDefs, copyright definitions
- CritDates, critical dates
- GlDefs, glossary definitions
- IBMProdInfo, IBM® product information
- IdxDefs, Index definitions
- LDescs, Link descriptions
- Owners
- ProdInfo, product informaiton
- PropDefs, property definitions
- QualifDefs, qualification definitions
- RevDefs, revision definitions

When you use these items in a division prolog; they take effect on that division and any nested divisions. To have these items affect the whole document, put them in the prolog. Here's a sample of a revision definition that affects this division; not the whole document:

```
<d>
<dprolog><titleblk>
<title>My Little Chapter</title>
</titleblk>
<revdefs>
<rev id="v3r4" ident="use">
<date>June 5th</date>
<desc>Something happened...</desc>
</rev>
</revdefs></dprolog>
<dbody>
<p>Here's the beginning of my chapter.</p>
<p rev="v3r4">Something that changed on June 5th.
</p>
</dbody></d>
```

Division introductions

You can introduce the division's content with the DIntro element. This element is optional; you should usually have your first paragraph of the DBody introduce the division's content. Anyway, Dintro follows the DProlog element. The next example shows the DIntro element.

```
<d>
<dprolog><titleblk>
<title>My Little Chapter</title>
</titleblk></dprolog>
<dintro>
<p>My little division introductory sentence.</p>
</dintro>
<dbody>
<p>Here's the beginning of my chapter.</p>
```

Partial table of contents

Division introductions can also create a partial table of contents for their corresponding chapter or part. In the part or division's introduction (DINTRO) tag, you code a table of contents (TOC) tag. This causes a partial table of contents to be generated at that point.

Here's the sample coding for a PTOC for a chapter:

```
<d>
<dprolog><titleblk>
<title>Sample chapter heading</title>
</titleblk></dprolog>
<dintro>
<toc><gendtitle></toc>
</dintro><dbody>
<d>
<dprolog><titleblk>
<title>Next heading</title>
</titleblk></dprolog>
<dbody></dbody></d>
<d>
<dprolog><titleblk>
<title>Another heading</title>
</titleblk></dprolog>
<dbody></dbody></d>
</dbody></d>
```

The partial table of contents lists the subordinate headings for the corresponding part or chapter.

Using parts to organize your chapters

IBMIDDoc includes the Part element, which you can use to divide your document into logical parts. This book has many divisions, but contains three parts (plus the appendixes). Parts do not affect the hierarchical ordering and numbering of divisions.

This example shows a sample book with 2 parts and 4 chapters:

```
<ibmiddoc>
<body>
<part>
<dprolog><titleblk>
<title>Introduction</title>
</titleblk></dprolog>
<dbody>
<d>
```

```
<dprolog><titleblk>
<title>Salads of our neighborhood</title>
</titleblk></dprolog>
<dbody></dbody></d>
<d>
<dprolog><titleblk>
<title>Salads of the world</title>
</titleblk></dprolog>
<dbody></dbody></d>
</dbody></part>
<part>
<dprolog><titleblk>
<title>Recipies</title>
</titleblk></dprolog>
<dbody>
<d>
<dprolog><titleblk>
<title>Egg salad</title>
</titleblk></dprolog>
<dbody></dbody></d>
<d>
<dprolog><titleblk>
<title>Tuna fish salad</title>
</titleblk></dprolog>
<dbody></dbody></d>
</dbody></part></body>
</ibmiddoc>
```

Starting page number control

The STARTPAGE (starting page) attribute allows you to assign the beginning page number to a section. You use that attribute on a division tag. It can be used with all first-level division tags. The STARTPAGE attribute value can be any positive integer, starting with 1. For example, if you use the following markup:

```
<d startpage="101"><dprolog><titleblk><title>Help information
...
<d startpage="201"><dprolog><titleblk><title>Safety information
```

the first chapter “Help information” starts on page 101, and the next chapter “Safety information” starts on page 201. Do not use the STARTPAGE attribute on the tags to create duplicate page numbers. Doing so can cause the wrong retrieval subject text to be associated with the page.

This can be used with the CHAPTERNUM attribute. For example, you could start formatting a document at chapter 21, page 83 with the following:

```
<d chapternum=21 startpage=83>
```

Chapter number control

You can use the CHAPTERNUM attribute on any first-level division tag to assign the chapter number. For example, this markup:

```
<d chapternum="13"><dprolog><titleblk><title>End of the line
```

would cause the chapter number for the “End of the line” chapter to be thirteen. In an appendix, the number becomes the corresponding letter. A ChapterNum of 4 would be appendix D.

This can be used with the STARTPAGE attribute. For example, you could start formatting a document at chapter 21, page 83 with the following:

```
<d chapternum=21 startpage=83>
```

For Options-by-IBM use: For example, if the French section needed to start on 1-37, the translator would create his section as follows:

```
<d chapternum=1 startpage=35>
<dprolog><titleblk>
<title>The title for this section -- goes in running foot</title>
</titleblk></dprolog>
<dbody>
<!-- The following bit skips two pages to start on page 37 -->
<p>&rbl;</p>
<?idd:page>
<p>&rbl;</p>
<?idd:page>
<!-- End of page skipping -->
<d>
<dprolog><titleblk>
<title>Translated title</title>
</titleblk></dprolog>
<dbody>
<p>Translated information</p>
.
.
.
</dbody></d>
</dbody></d>
```

If the translated section is supposed to start on an even page, the STARTPAGE would be one less than the first desired page, and just one blank page added before the level-2 heading section. When composing, you can specify /SHEET:2 (for starting on an even page) or /SHEET:3 (for starting on an odd page) to create the PostScript file without the dummy starting page. The dummy page could also be stripped when combining the files.

Changing column layouts

Normally, your document's style determines the number of columns, also called the layout, of your document. You can, if you want, change from one column layout to another.

You can override this layout for your document by using the layout attribute on the IBMIDDoc tag.

You can also use the layout attribute on Division tags (and other major headings such as Preface) to override the layout style for that chapter, appendix, or magor heading.

You can pick from the following values; not all tags support all these values:

LAYOUT=Default-Layout | OneCol | TwoCol | ThreeCol
Specifies the column-style for this portion of the book.

Default-Layout
The section uses the default layout for the document style.

OneCol
The entries format across the entire page.

TwoCol
The entries format in two columns.

ThreeCol
The entries format in three columns.

Creating an information architecture

You can use the Metadata tag to classify the type, audience, and task information for a topic. This helps search programs and other programs find, filter, or select information. This is passed through to the XHTML output as metadata keywords.

For example, here is an example of a division that has a Metadata tag; the tag classifies the information:

```
<d id="feederinst">
<dprolog><titleblk>
<title>Installing your Fruit-Bat Feeder</title>
</titleblk>
<metadata type="task" job="installing"
 audience="user" experiencelevel="general">
</dprolog>
...
```

Subordinate topics inherit the metadata classifications from their parent topics.

See “MetaData (information architecture)” on page 343 for a description of the tag and the attributes allowed.

See “Architected online information and Information Centers (vs books)” for tips on creating architected information.

Architected online information and Information Centers (vs books)

Notes from an XML workgroup conference call; presented Sept. 4/2001 by Leigh Davidson

Here are tips for creating architected online information; these are not books. The information here was usde by several sites that create Information Centers.

- The basic unit of information is a topic, not chapter or section. A topic is a division in IBMIDDoc.
- Organization: The information is a hyperlinked web, not a linear presentation.
- Entry points to the information includes the following:
  - Search hit list
  - Links from other topics
  - Hierarchical navigation frame
  - Any combination/permutation of these techniques

What is a topic?

- Granular piece of information
- Probably “information-typed” (more on this below)
- Some examples:
  - One or two paragraphs to explain a concept
  - A procedure consisting of a half-dozen steps
  - The syntax of a single command
  - Description of a single user interfacecontrol
- Should usually fit into one screen with no scrolling

Information types:

- Refer to UA Central: http://ua.raleigh.ibm.com/ua
- Categories of information
- Examples:
  - Task information provides procedural details such as step-by-step instructions.
  - Concept information provides background information that users need to know before they can successfully work with a product or interface.
  - Reference information provides quick access to facts, but no explanation of concepts or procedures. It is usually assumed that users already understand the base technology.
- Other possible types or subtypes: context-sensitive help, sample, tutorial, troubleshooting, and so forth.
- Fundamental principle: separate information according to type

Recommended approach:
- Write task-oriented information; not function-oriented information
- Use task analysis to organize your task list
- Write task topics
- As you go, identify prerequisite concepts and other supporting information (reference, tutorials, samples, glossary entries, etc.)
- Design the linking structure
- Never stop thinking about your users:
  - What tasks will they want or need to perform?
  - What does "completeness" mean to them? Value their time; don't bog them down with too much information.
  - What might they want or need to know next? Don't add your list of related links as an afterthought.

Challenges:
- Write **less**.
- Retrievability
- Consistency - organization, linking techniques, and chunking should all follow recognizable patterns (as well as style and tone)

Chapter 3. All kinds of lists

Several types of list elements are explained in this chapter, including:

- UL, unordered (see "Unordered lists") and simple (see "Simple lists" on page 30)
- OL, ordered (see "Ordered lists" on page 30)
- DL, definition (see "Definition lists" on page 32)
- ParmL, parameter (see "Parameter lists" on page 34)
- MsgList, message (see "Message and code lists" on page 38)

In addition, we discuss other things that are often used in lists:

1. List items can be compacted (see "Compacting lists" on page 36)
2. LiBlk, list item block; these allow you to group related list items into information blocks (see "Grouping list items" on page 37)
3. Bridges; these are for transitions between list items (see "Separating or bridging list items" on page 38)

Unordered lists

Unordered lists are used when the items in the list are fairly long, maybe even many paragraphs, but you don't want to imply any particular sequence (as you would with an ordered list). The default appearance for an unordered list is as a bulleted list. Here's an example of an unordered list:

- This is an item in an unordered list. To separate it from other items in the list, the formatter puts a bullet beside it.
- The paragraph that is contained in the LI element is part of the list item which contains it.

  This is the contained paragraph.
- This is a separate list item in our unordered list.

Here is the IBMIDDoc markup for the unordered list in the previous example.

```
<ul>
<li>This is an item in an unordered list. To separate
it from other items in the list, the formatter puts
a bullet beside it.</li>
<li>The paragraph that is contained in the LI element
is part of the list item which contains it. <p>This
is the contained paragraph.</p></li>
<li>This is a separate list item in our unordered
list.</li>
</ul>
```

Many IBMIDDoc elements can contain lists. If you do not want the list to be contained in the element that precedes it, be sure to end the preceding element before starting the list element.

The example that follows illustrates an unordered list that is not contained by the paragraph element that immediately precedes it.

```
<p>
Text of paragraph.
</p>
<ul>
<li>Abbrev</li>
```

```
<li>Abstract</li>
<li>Bibliog</li>
<li>Appendix</li>
<li>Glossary</li>
</ul>
```

You can also use ULTYPE=CHECKOFF on an unordered list to create an unordered checkoff list. For example, here's an unordered, checkoff list:

__ • Abbrev

__ • Abstract

__ • Bibliog

__ • Appendix

__ • Glossary

Simple lists

Simple lists are just what you'd think they are; they have no dingbat.[1] For example, here's a simple list:

bread

butter

cheese

bananas

A simple list starts with an unordered list, then you set the style attribute to "simple":

```
<ul ultype="simple">
<li>bread</li>
<li>butter</li>
<li>cheese</li>
<li>bananas</li>
</ul>
```

You can also use ULTYPE=SimpleCheckoff on an unordered list to create a simple checkoff list. For example, here's a simple, checkoff list:

__ bread

__ butter

__ cheese

__ bananas

Ordered lists

An ordered list contains information that must be listed in a specific sequence. This is often a list of steps which must be performed in a certain order, such as a recipe or tearing apart a PC. An ordered list looks like this:

1. Cream butter and sugar together until fluffy.
2. Beat in egg yolks one at a time.
3. Add nutmeg, cinnamon, and vanilla, and mix thoroughly. The batter should be smooth and glossy and stream off the spoon in ribbons.
4. Fold in beaten egg whites.

1. dingbat. (1) In printing: Any typographical ornament not further specified. (2) In old TV Shows: What Archie Bunker would call his wife, Edith.

Do not overmix; the batter should be light and fluffy.

(I'm getting hungry.) The corresponding IBMIDDoc markup is as follows:

```
<ol>
<li>Cream butter and sugar together until fluffy.</li>
<li>Beat in egg yolks one at a time.</li>
<li>Add nutmeg, cinnamon, and vanilla, and mix thoroughly.
The batter should be smooth and glossy and stream
off the spoon in ribbons.</li>
<li>Fold in beaten egg whites.
<p>Do not overmix; the batter should be light and fluffy.</p></li>
</ol>
```

Checkoff ordered lists

Sometimes when describing procedures, you need to give your reader a checkoff space as an aid to ensure that they perform every step. The checkoff list is an ordered list; you get it adding the OLTYPE=CHECKOFF attribute to your OL tag. For example:

__ 1. Verify that air pressure is in normal range.

__ 2. Verify that fuel level is in safe zone.

__ 3. Verify that water valve is in open position.

Here's the coding:

```
<ol oltype="checkoff">
<li>Verify that air pressure is in normal range.</li>
<li>Verify that fuel level is in safe zone.</li>
<li>Verify that water valve is in open position.</li>
</ol>
```

Customer setup lists

Customer setup lists use the word "Step" in the list item, together with the number. These lists also use the OLTYPE attribute on the ordered list tag (OL), with a value of STEP. For example:

Step 1. Open the carton.

Step 2. Remove the top layer of packing material.

Step 3. Take the stuff out of the box.

Step 4. Give the box to the kids to play with, while you proceed with the next step.

Here's its coding:

```
<ol oltype="step">
<li>Open the carton.</li>
<li>Remove the top layer of packing material.</li>
<li>Take the stuff out of the box.</li>
<li>Give the box to the kids to play with,
while you proceed with the next step.</li>
</ol>
```

You can also have checkoff-setup lists. These lists also use the OLTYPE attribute with a value of CHECKOFFSTEP. For example:

__ Step 1. Fold along dotted line C.

__ Step 2. Insert tab B into slot A.

__ Step 3. Throw these instructions out the window.

Here's its coding:

```
<ol oltype="checkoffstep">
<li>Fold along dotted line C.</li>
<li>Insert tab B into slot A.</li>
<li>Throw these instructions out the window.</li>
</ol>
```

Continuing ordered lists

Sometimes, you may have lists that need to continue around table cells or even from division to division. The ordered list (OL) tag has the attributes SEQ, ID, and SEQID to allow you to have an ordered list continue from where a previous list left off. For example, here's a continued checkoff list :

Formatted Example

__ 1. Open the carton.

__ 2. Remove the top layer of packing material.

__ 3. Take the stuff out of the box.

The first list is ended — honest, this is not a bridge. The list then continues:

__ 4. Fold along dotted line C.

__ 5. Insert tab B into slot A.

End of Formatted Example

Here's its markup:

```
<ol seq="start" oltype="checkoff" id="swingsets">
<li>Open the carton.</li>
<li>Remove the top layer of packing material.</li>
<li>Take the stuff out of the box.</li>
</ol>
<p>The first list is ended — honest, this is
not a bridge. The list then continues:</p>
<ol seqid="swingsets" seq="end" oltype="checkoff">
<li>Fold along dotted line C.</li>
<li>Insert tab B into slot A.</li>
</ol>
```

Definition lists

Definition lists are a particular kind of list you can use when you want to pair a term or phrase with a description of it.

Here's a formatted example of a definition list:

gopher	A burrowing rodent that feeds on roots of plants.
lawn	Gopher highway. Can be identified by dinner-plate-sized mounds of dirt where grass used to be.
agapanthus	Lovely flowering plant, the roots of which are the preferred food of gophers. If your flourishing agapanthus suddenly keels over, it means a gopher has had a feast.

Here's its coding:

```
<dl>
<dlentry><term>gopher</term>
<defn>A burrowing rodent that feeds on roots of plants.
</defn>
</dlentry>
<dlentry><term>lawn</term>
<defn>Gopher highway. <p>Can be identified by dinner-plate-sized
mounds of dirt where grass used to be.</p></defn>
</dlentry>
<dlentry><term>agapanthus</term>
<defn>Lovely flowering plant, the roots of which are
the preferred food of gophers. <p>If your flourishing
agapanthus suddenly keels over, it means a gopher
has had a feast.</p></defn>
</dlentry>
</dl>
```

You can use the TERMWIDTH attribute to determine the indentation size of the definition list. The valid choices are: small (.5 inch, the default), medium (1 inch), large (2 inches), and 1 (1-character) and 2 (2-characters).

Small Here's a sample of the small setting.

Medium Here's a sample of the medium setting.

Large Here's a sample of the large setting.

1 Here's a sample of the 1-character setting.

2 Here's a sample of the 2-character setting.

Definition lists can also have headings; for example:

Setting Description

Low A good setting for simmering soups.

Medium
After the water has boiled, use this setting for cooking the spaghetti.

High Use this setting to get water boiling fast.

This is done with the TermHd and DefnHd tags. Here's the source:

```
<dl><termhd>Setting</termhd>
<defnhd>Description</defnhd>
<dlentry><term>Low</term>
<defn>A good setting for simmering soups.</defn>
</dlentry>
<dlentry><term>Medium</term>
<defn>After the water has boiled, use this setting
for cooking the spaghetti.</defn>
</dlentry>
<dlentry><term>High</term>
<defn>Use this setting to get water boiling fast.
</defn>
</dlentry>
</dl>
```

If you want to change your list heading and term style, you can use the TERMSTYLE and HEADSTYLE attributes to set a different highlight. For example:

Animal
Description

Cat A house pet that purrs when happy.

<u>Dog</u>	A house pet that wags its tail when happy.

Here's the source:

```
<dl termstyle="bold-italic-underlined" headstyle="bold-italic-h">
<termhd>Animal</termhd>
<defnhd>Description</defnhd>
<dlentry><term>Cat</term>
<defn>A house pet that purrs when happy.</defn>
</dlentry>
<dlentry><term>Dog</term>
<defn>A house pet that wags its tail when happy.</defn>
</dlentry>
</dl>
```

You can also group terms together using the DLBlk (definition block) tag. You can use DLBlk to create logical groups within a long definition list, or use two DLBlks with a Bridge between them to highlight some relationship between groups of entries. For example:

Cat	A house pet that purrs when happy.
Dog	A house pet that wags its tail when happy.
Fish	A house pet with scales that swims.
Turtle	A house pet with scales that swims and walks slowly.

Here's the source:

```
<dl>
<dlblk>
<dlentry><term>Cat</term>
<defn>A house pet
that purrs when happy.</defn></dlentry>
<dlentry><term>Dog</term>
<defn>A house pet that wags
its tail when happy.</defn></dlentry>
</dlblk>
<dlblk>
<dlentry><term>Fish</term>
<defn>A house pet
with scales that swims.</defn></dlentry>
<dlentry><term>Turtle</term>
<defn>A house pet with
scales that swims and walks slowly.</defn></dlentry>
</dlblk>
</dl>
```

Parameter lists

Parameter lists are used in programming documentation when you have to explain the elements of the programming syntax. Here's an example of a parameter list:

KEYWORD = <u>DEFAULT</u>|VALUE
This is the description of the parameter above. It could go on for many pages, if necessary. (Of course, that means we have a very complicated parameter to describe.)

KEYWORD2 = {ABC|XYZ}
[KEYWORD3 = GGG]
This description applies to the two parameters above. Often in examples of programming syntax, it is necessary to use symbols for the brackets and braces.

KEYWORD3
  Here's a term that uses the syntax phrase (SYNPH); it allows you to use the same items as a syntax diagram.

Parameter lists are similar to definition lists. They involve three elements: ParmL (parameter list), Term, and Defn (definition). Term and Defn are used with other elements for the same function, including glossary and definition lists, among others. When doing parameter lists for programming syntax, you will also need to use these elements:

- PK (programming keyword)
- PV (programming variable)
- SYNPH (syntax phrase), with KWD (keyword), VAR (variable), and DELIM (delimiter)

The Term elements assume that what you are entering is a required or optional programming keyword. For a default programming keyword or programming variable, edit the attributes for these PK or PV elements, and set the OPTREQ attribute value to `DEF`.

The IBMIDDoc markup for the example parameter list is shown in the example that follows.

```
<parml>
<parm><term>KEYWORD = <pk optreq="DEF">DEFAULT</pk>|VALUE
</term>
<defn>This is the description of the parameter above.
It could go on for many pages, if necessary. (Of course,
that means we have a very complicated parameter to
describe.)</defn>
</parm>
<parm><term>KEYWORD2 = &lbrc;ABC|XYZ&rbrc;</term>
<term>&lbrk;KEYWORD3 = GGG&rbrk;</term>
<defn>This description applies to the two parameters
above. Often in examples of programming syntax, it
is necessary to use symbols for the brackets and braces.
</defn>
</parm>
<parm><term><synph><kwd>KEYWORD3</kwd></synph></term>
<defn>Here's a term that uses the syntax phrase (SYNPH);
it allows you to use the same items as a syntax diagram.
</defn>
</parm>
</parml>
```

You can use the TERMWIDTH attribute to determine the indentation size of the parameter list. The valid choices are: small, medium, and large. You can use the TERMWIDTH attribute to determine the indentation size of the definition list. The valid choices are: small (.25 inch, the default), medium (.5 inch), large (1 inch), and 1 (1-character), and 2 (2-character).

Small
  Here's a sample of the small setting.

Medium
  Here's a sample of the medium setting.

Large Here's a sample of the large setting.

1 Here's a sample of the 1-character setting.

2 Here's a sample of the 2-character setting.

If you want to change your list heading and term styles , you can use the TERMSTYLE and HEADSTYLE attributes to set a different highlight. For example:

Animal
: ***Description***

Cat
: A house pet that purrs when happy.

Dog
: A house pet that wags its tail when happy.

Here's the source:

```
<parml termstyle="bold-italic" headstyle="bold-italic-underlined-h">
<termhd>Animal</termhd>
<defnhd>Description</defnhd>
<parm><term>Cat</term>
<defn>A house pet that purrs when happy.</defn>
</parm>
<parm><term>Dog</term>
<defn>A house pet that wags its tail when happy.</defn>
</parm>
</parml>
```

Compacting lists

Compact specifies that you do not want a blank line between each list item. Compact applies only to the space between list items and not to any space between paragraphs within a list item. Compact lists are only available in Xyvision, BookMaster, HTML, and IPF. Documents written in Xyvision automatically have a half a line space.

To specify a list as compact, use the LineSpace attribute. For example:

```
<ul linespace="compact">
```

The lists you can compact are: DL, OL, UL, GL, Msglist, Parml, and Notelist.

Scaling list dingbats

> **dingbat.** (1) A small object, such as a stick or stone, suitable for hurling at another object. (2) Any unspecified gadget or other article. (3) Printing Any typographical ornament not further specified. (4) Archie Bunker's "pet" name for his wife, the lovable Edith Bunker, from the 1970s TV series *All in the Family*.

We are, of course, interested only in the third of these definitions, fascinating as the others are to contemplate.

For our purposes, given an unordered list

- that looks
- like this

the dingbat is the • that signals each list item. In an ordered list

1. that looks
2. like this

our dingbats are "1" and "2" — the numbers.

For unordered and ordered lists, you can specify a scaling factor (with DBSCALE). With the scaling factor, you can make the dingbat bigger or smaller. We can make each list fancier:

1. that looks
2. like this

The dbscale is 200 — twice the size of normal:

```
<ol dbscalepct="200">
<li>that looks</li>
<li>like this </li>
</ol>
```

And for the unordered list:

- that looks
- like this

The dbscale is 50 — half the size of normal:

```
<ul dbscalepct="50">
<li>that looks </li>
<li>like this</li>
</ul>
```

The dbscale only applies to the list on which is it specified. It does not apply or inherit from outer lists to inner lists.

Grouping list items

You can use list item block (LIBlk, DLBlk, and ParmBlk) elements to contain groups of similar items. LIBlk is for list items, DLBlk is definition list items, and ParmBlk is for parameter list items. In the example that follows, hardware and software are grouped into blocks of list items.

For example:

1. 1 GIG SCSI-2 Hard Disk
2. 32 MB RAM
3. 128-Bit 8MB VRAM Video
4. 21-Inch Monitor
5. Great Word Processor
6. Best Multimedia App
7. Voice Mail

Here’s its markup:

```
<ol>
<liblk>
<li>1 GIG SCSI-2 Hard Disk</li>
<li>32 MB RAM</li>
<li>128-Bit 8MB VRAM Video</li>
<li>21-Inch Monitor</li>
</liblk>
<liblk>
<li>Great Word Processor</li>
<li>Best Multimedia App</li>
<li>Voice Mail</li>
</liblk>
</ol>
```

Processing Note

For Xyvision processing, if you want to have all the text in the LIBLK kept on the same page, use the attribute `style="xpp:(keep)"`. Be cautious when using this feature. If the text does not fit on a page, you may get a formatting error. Remove the style or shorten the content to have the pages print correctly.

Separating or bridging list items

Sometimes in an ordered list you want to break the list for some explanatory material and then resume the numbering where you left off. You do this with the Bridge element.

Suppose you wanted to do this:

1. Saute the shallots and chopped mushrooms until the shallots are tender and the liquid from the mushrooms has cooked away.
2. Brown the sausage and add to the mushroom mixture.

The above may be prepared several hours in advance and refrigerated. Then, 30 minutes before serving time, finish the dish

3. Mix one can of tomato sauce with the mushroom and sausage mixture and bring to a slow simmer.
4. Add the heavy cream and immediately pour into a casserole.
5. Pop into 350-degree oven for 15 minutes.

Here's its markup:

```
<ol>
<li>Saute the shallots and chopped mushrooms until
the shallots are tender and the liquid from the mushrooms
has cooked away.</li>
<li>Brown the sausage and add to the mushroom mixture.
</li>
<bridge><p>The above may be prepared several hours in
advance and refrigerated. Then, 30 minutes before
serving time, finish the dish</p></bridge>
<li>Mix one can of tomato sauce with the mushroom
and sausage mixture and bring to a slow simmer.
</li>
<li>Add the heavy cream and immediately pour into
a casserole.</li>
<li>Pop into 350-degree oven for 15 minutes.</li>
</ol>
```

Message and code lists

If you don't have any messages or codes to worry about, just skip this section and go on.

Both message and code lists use the MsgList element. A code list documents numeric values that have specific meaning, for example error codes. A message list documents text messages, which may each have a message number. Each entry in a message or code list is contained in a MSG element.

For the code list, use the Code element for the numeric value. For the message list, use a MsgNum and MsgText element. If the message has no number, use just the

MsgText element. If the message text has a variable in it, you use the MV element for the message variable. Use the MV element in both the message text and in any descriptive text.

After you have entered the code or the message number and the text, you use the MsgItem element to contain the information about the message, according to the class of the information. IBMIDDoc has a number of predefined classes of message item information. These pre-defined classes have a generated message subheading. These classes are:

Class	**Default subheading text**
author-defined	Author defined class using the MsgItemDef element (currently not supported by the output formatters).
DEST	Destination
XPL or EXPLANATION	Explanation
MODULE	Module
NUMBYTES	Number of Error Bytes
ORESP	Operator Response
PRESP	Programmer Response
PROBD	Problem Determination
SEVERITY	Severity
SPRESP	System Programmer Response
SYSACT	System Action
URESP	User Response

This is a sample message list:

DJI7832E **This message is issued when no data set of the name** ***file-name*** **is found.**

Explanation: The processor could not locate the data set named *file-name*.

Severity: 8

Problem Determination: You would appear to have a problem.

User Response: Search high and low for the data set.

This message has no number

Explanation: This message has no message number; only text. These are really insidious because it makes finding the message very hard.

Here is the markup:

```
<msglist>
<msg>
<msgnum>DJI7832E</msgnum>
<msgtext>This message is issued when no data set of
the name <mv>file-name</mv> is found.</msgtext>
<msgitem class="xpl">
<p>The processor could not locate the data set named <mv>
file-name</mv>.</p>
</msgitem>
<msgitem class="severity">
<p>8</p>
</msgitem>
<msgitem class="probd">
<p>You would appear to have a problem.</p>
</msgitem>
<msgitem class="uresp">
<p>Search high and low for the data set.</p>
</msgitem>
</msg>
<msg>
<msgtext>This message has no number</msgtext>
<msgitem class="xpl">
```

```
<p>This message has no message number; only text.
These are really insidious because it makes finding
the message very hard.</p>
</msgitem>
</msg>
</msglist>
```

Code lists are just the same, except that you use the Code element instead of MsgNum and MsgText. Code lists still use MsgItem elements. Here is a short code list:

123

Explanation: This is a simple code.

Here's its coding:

```
<msglist>
<msg><code>123</code>
<msgitem class="xpl">
<p>This is a simple code.</p>
</msgitem>
</msg>
</msglist>
```

Overriding the message list subheadings

The text generated in association with MsgItem classes (XPL, URESP, and the rest) is not always suitable to every type of document containing these lists. For instance, you might want to use the subheading "Cause" or "Reason" instead of "Explanation", and "Recovery" instead of "User Response". Some people prefer "Severity Code" to "Severity". Some places don't have system programmers, but do have administrators or supervisors.

To meet all of these requirements and still preserve some order in the chaos, the MsgItemDef element can be used to override the default text that corresponds to the MsgItem classes. With these attributes you can specify the text that you want printed when the tag is used.

Here is a sample message list with changed headings. The heading are overridden for this list only; because the MsgItemDef tags are within the message list.

A12 **Closet full: Insufficient storage to proceed.**

Why: There are too many clothes in the closet.

What to do: Remove some clothes from the closet and restart.

Here is its coding:

```
<msglist>
<msgitemdef classname="xpl"><title>Why</title></msgitemdef>
<msgitemdef classname="uresp"><title>What to do</title>
</msgitemdef>
<msg>
<msgnum>A12</msgnum>
<msgtext>Closet full: Insufficient storage to proceed.
</msgtext>
<msgitem class="xpl">
<p>There are too many clothes in the closet.</p>
</msgitem>
<msgitem class="uresp">
<p>Remove some clothes from the closet and restart.
```

```
</p>
</msgitem>
</msg>
</msglist>
```

Note that when you use the class attribute, you should pick a value that is close in intent to the original meaning of the tag. That is, if you want to print “Reason”, use the XPL class; don’t use one — for instance, NUMBYTES — that is totally unrelated to the text you are printing. Do this because you may have future applications for this text, such as extracting from a text data base all messages and their explanations. If you don’t preserve the meaning of these tags, these future applications won’t be possible.

To globally override the message headings, use the MsgItemDef tags in the document prolog. For example:

```
<prolog>
 ...
<propdefs>
<msgitemdef classname="xpl"><title>Why</title></msgitemdef>
<msgitemdef classname="uresp"><title>What to do</title>
</msgitemdef>
</propdefs>
 ...
</prolog>
 ...
<msglist>
<msg>
<msgnum>A12</msgnum>
<msgtext>Closet full: Insufficient storage to proceed.
</msgtext>
<msgitem class="xpl">
<p>There are too many clothes in the closet.</p>
</msgitem>
<msgitem class="uresp">
<p>Remove some clothes from the closet and restart.
</p>
</msgitem>
</msg>
</msglist>
```

Chapter 4. Highlighting, Citing, Noting, and Quoting

IBMIDDoc provides several different ways to highlight text and contains many ways to create notes, annotations, and footnotes. This chapter introduces each of these.

These element types include:
- Phrases (see "Highlighting")
- Citations (see "Simple title citations" on page 45)
- Notes (see "Notes" on page 45)
- Note lists (see "Note lists" on page 46)
- Footnotes (see "Footnotes" on page 46)
- Quotes and long quotes (see "Quotes and excerpts" on page 47)
- The Perils of Processing (see "The perils of processing: Attention, caution, and danger" on page 48)
- Labeled boxes (see "Labeled boxes" on page 48)
- Author notes (see "Annotations" on page 49)
- Trademarks (see "Trademarks" on page 50)
- Qualifications (see "Qualifying information" on page 49)

This chapter also describes how to create style classes for elements that can be passed through to XHTML documents. See "Using document classes with XHTML style sheets" on page 51 for that information.

Highlighting

You've already seen many examples in this book of things that are highlighted. By highlighting, we mean emphasizing the text by setting it in a different font or perhaps underscoring it. Many phrase styles are supported. They are mapped as <PH style=attribute>. The style attribute values include:
- base
- **bold**
- *italic*
- ***bold italic***
- underlined
- $^{\text{superscript}}$
- $_{\text{subscript}}$
- `monospaced`
- SMALLCAPS. Note that YOU need to do the uppercase conversion yourself. This is because not all languages do proper uppercase conversion of lowercase letters.
- **underlined bold**
- *underlined italic*
- ***underlined bold italic***
- UNDERLINED SMALLCAPS

These values are only valid on the PH tag's style attribute. Here's a sample:

Formatted Example

Hey there! This is **very** **important**! Don't go out in the *rain* ***without your galoshes***!

End of Formatted Example

Here's its markup:

```
<ph style="Bold Italic">Hey there!</ph>
This is <ph style="Underlined Bold">very</ph>
<ph style="Bold">important</ph>! Don't go out in the
<ph style="Italic">rain</ph> <ph style="Underlined Bold Italic">
without your galoshes</ph>!
```

Other elements included for denoting values and phrases are shown in Table 1.

Table 1. Phrase types

Description	Markup	Result
APL	`<apl>APL</apl>`	*APL*
binary (bin)	`<bin>0101</bin>`	B'0101'
character (char)	`<char>character</char>`	"character"
decimal (dec)	`<dec>123</dec>`	123
example phrase (xph)	`<xph>example</xph>`	`example`
hexadecimal (hex)	`<hex>10FE</hex>`	X'10FE'
marked deletion (md)	`<md rev="rev1">marked deletion</md>`	~~marked deletion~~
message variable (mv)	`<mv>message variable</mv>`	*message variable*
number with specified base (num)	`<num base="3">120</num>`	120
octal (oct)	`<oct>013 736</oct>`	O'013 736'
programming keyword	`<pk>keyword</pk>`	**keyword**
programming keyword default	`<pk optreq="def">default</pk>`	**default**
programming variable	`<pv>keyword</pv>`	*keyword*
reference key (refkey)	`<refkey>reference key</refkey>`	reference key
term	`<term>term</term>`	term

You can also nest highlighted phrases — that is, put one kind inside another. The formatter causes the nested highlighting to inherit the highlighting of the previous style. In the following example, the italic and underlined text are also bold because the whole sentence is marked bold:

Speak *softly* and carry a BIG stick.

Here's its markup:

```
<ph style="Bold">Speak <ph style="Italic">softly</ph>
and carry a <ph style="Underlined">BIG stick
</ph></ph>.
```

Simple title citations

While IBMIDDoc provides extensive bibliographic markup (see Chapter 14, "Bibliographies and citations" on page 141), sometimes you just need a simple inline title citation. For this you use the CIT element (and some others). For example, here's a reference to a non-IBM book:

Formatted Example

Huckleberry Finn, by Mark Twain, is a most excellent book.

End of Formatted Example

Here's its markup:

```
<cit><bibentry><doctitle><titleblk><title>Huckleberry
Finn</title></titleblk></doctitle></bibentry></cit>,
by Mark Twain, is a most excellent book.
```

Here's a reference to an IBM book:

Formatted Example

The *BookMaster User's Guide* is the book to emulate.

End of Formatted Example

Here's its markup:

```
The <cit><ibmbibentry><doctitle><titleblk><title>
BookMaster User's Guide</title></titleblk></doctitle>
</ibmbibentry></cit> is the book to emulate.
```

Notes

The Note element contains a single note. For example:

Note: Thinking of a seashore, green meadow, or cool mountain overlook can help you to relax and be more patient.

Here's its coding:

```
<note><notebody>Thinking of a seashore, green meadow,
or cool mountain overlook can help you to relax and
be more patient.</notebody></note>
```

If you want the word "Note" to be something else, use the TITLE element; for example:

Tip: Don't sit under the apple tree with anyone else but me.

Here is its coding:

```
<note><title>Tip</title>
<notebody>Don't sit under the apple tree with anyone else but me.
</notebody>
</note>
```

Note lists

The NoteList element contains an ordered list of notes. The list is ordered because there is usually a priority to the notes. For example:

Notes:

1. Make a To Do list
2. Prioritize sensibly
3. Avoid interruptions where possible
4. Check on your progress toward monthly goals
5. Plan for the next work week
6. Do something for the fun of it
7. Spend some quality time with your pet

Here's its coding:

```
<notelist>
<li>Make a To Do list</li>
<li>Prioritize sensibly</li>
<li>Avoid interruptions where possible</li>
<li>Check on your progress toward monthly goals</li>
<li>Plan for the next work week</li>
<li>Do something for the fun of it</li>
<li>Spend some quality time with your pet</li>
</notelist>
```

Groups of notes can be organized into blocks using the LIBlk element.

You can also use your own title instead of "Notes" by adding a Title. For example:

Watch out for these:

1. things that go bump in the night
2. green eggs and ham

Here's its markup:

```
<notelist><title>Watch out for these</title>
<li>things that go bump in the night</li>
<li>green eggs and ham</li>
</notelist>
```

You can compact, bridge, and group list items together. This is done in the same way as ordered lists; see Chapter 3, "All kinds of lists" on page 29.

Footnotes

The FN element is used to annotate text with notes that are part of the narrative content of the document, but that are not appropriate for inclusion inline with the document text. The information contained in the FN element is associated with its containing element.

Formatted Example

Here’s a footnote[2] around here somewhere.

End of Formatted Example

Here’s its coding:

```
<p>There's a footnote<fn>While some folks do not like
footnotes; they sometimes contain a nugget of priceless
lore. Did you know IBMIDDoc's grandmother was named
ISIL?</fn> around here somewhere.</p>
```

You can also define a footnote and use it in multiple places. First, define the footnote (FN tag) and give it an ID attribute. Then, use the FN tag with a REFID attribute to refer to that footnote.

Formatted Example

Here’s a sentence[3] that uses several footnotes[3].

End of Formatted Example

Here’s its coding:

```
<p>
<fn id="multiple">Here's another little footnote.</fn>Here's
a sentence<fn refid="multiple"> that uses several
footnotes<fn refid="multiple">.</p>
```

Quotes and excerpts

IBMIDDoc provides for two different kinds of quotations — inline quotations and excerpts (which we call “long quotations” although it really has nothing to do with the length). For simple inline quotations, use the Q element. These can be nested. For example, this contains two quotes, one inside the other. The formatter knows to automatically switch from double quotes to single quotes (in hardcopy anyway).

Formatted Example

George said; “She said ‘Yes!’ to me.”

End of Formatted Example

Here’s its coding:

```
George said; <q>She said <q>Yes!</q> to me.</q>
```

For long quotations or excerpts, use the LQ element. Who remembers this quote from History class:

> The only thing we have to fear is fear itself.

Here’s its coding:

2. While some folks do not like footnotes; they sometimes contain a nugget of priceless lore. Did you know IBMIDDoc’s grandmother was named ISIL?

3. Here’s another little footnote.

```
<lq>The only thing we have to fear is fear itself.
</lq>
```

Labeled boxes

Labeled boxes are a special style of paragraph block. For example:

Here’s my cute little box

Here’s something that I’m really proud of.

Here’s its coding:

```
<pblk style="lblbox">
<title>Here's my cute little box</title>
<p>Here's something that I'm really proud of.</p>
</pblk>
```

Beware of over-using these; and of trying to put too much information into them.

The perils of processing: Attention, caution, and danger

These elements are used to contain information about situations that can dangerous to people, equipment, or data.

Attention
Use an Attention notice to indicate the possibility of damage to a program, device, system, or data.

Warning
Use a Warning notice to indicate the possibility of damage to a program, device, system, or data.

Caution
Use a Caution notice to call attention to a situation that is potentially hazardous to people because of some existing condition. For example, you might use a Caution notice to warn about the hazard of paper cuts when someone opens a fresh ream of paper.

Danger
Use a Danger notice to call attention to a situation that is potentially lethal or extremely hazardous to people. For example, after a computer side panel is removed, exposed high-voltage wires might be lethal.

Attention: Here’s a way to get someone’s attention.

CAUTION:
Watch out for these!

DANGER

Really watch out for these!

Here are their codings:

```
<attention>Here's a way to get someone's attention.</attention>
<caution>Watch out for these!</caution>
<danger>Really watch out for these!</danger>
```

Annotations

The Annot element is used to contain comments about the content of its containing element. These comments can be notes to reviewers, other writers, editors, vendors, and so forth. Annotations do not contain comments you want to appear in a final draft of your document. Annotation content is not part of the narrative text of your document.

Annot has NO formatting associated with it. All formatting comes from the markup within the annotation body.

Processing Note

The Xyvision formatter does not print annotations under any circumstances. BookMaster hides the content by default, unless you specify this runtime option:

```
sysvar(A yes)
```

Migration Note

Do not use the Annot element to comment out information. Because Annot is an element in the document hierarchy, it cannot span structures in the document.

If you want to selectively print annotations, each Annot element should be contained in a marked section or use a PROPS value as shown below.

```
<P>Remove the cover from the system unit by unscrewing the tabs on the
rear of the unit.
<ANNOT PROPS="hide">
<TITLE>>System Test Note</TITLE>
<ANNOTBODY>
<P>
Please advise us of any discrepancies in the installation
instructions in this section.
</P>
</ANNOTBODY>
</ANNOT>
</P>
```

Migration Note

Any ANNOT tags used to comment out information in Bookmaster will be presented within a labeled box. You can delete these or convert them to SGML comments.

Qualifying information

When you have to qualify information as applying to a particular product or system in a book about multiple products or systems, there are several techniques you can use. One, of course, is simply to say, “If you are using Model 9, then....”. Another method, for major differences, is to put some qualification in the section heading. Still another method is to use a formatting convention such as the one provided by the QualifDef (quilifying information definition) element and the QUALIF (qualifying information) attribute. This is suitable only for qualification at the level of a paragraph or list item or greater. It is not suitable for single-phrase qualifications, or for qualifications of many pages.

You begin by defining your qualification in the document's prolog, using the QualifDef and Qualif elements. For example, this specifies two qualifications, one for Windows® 99 and another for OS/2.5:

```
<qualifdefs>
 <qualif id="win99" ident="use">
  <title>Windows/99</title>
  <desc>Windows/99 information</desc>
 </qualif>
 <qualif id="os25" ident="use">
  <title>OS/2.5</title>
  <desc>OS/2.5 information</desc>
 </qualif>
</qualifdefs>
```

Here are sample paragraphs for each type of qualification:

Windows/99

This operating system is great for home use.

End of Windows/99

OS/2.5

This operating system is the business-oriented, industrial-strength operating system.

End of OS/2.5

Here is their markup:

```
<p qualif="win99">This operating system is great for
home use.</p>
<p qualif="os25">This operating system
is the business-oriented, industrial-strength operating system.</p>
```

QUALIFs cannot be nested. Also, the text must be short enough to fit within the column along with the words "end of," plus the blanks, corners, and at least a little of the line. This can be a tight fit in a double-column layout. If you use the Qualif inside a boxed figure or a ruled table, the corners may overlay the rules.

Trademarks

The TM tag identifies the trademark terms in your source by surrounding the trademark term or phrase. This tag has attributes that are not translated; they contain no "MRI". The TM attributes contain information for the author. The TMType attribute creates the appropriate trademarking character after the term or phrase.

The ID Workbench files IDDIRTM.LST or IDTMSCAN.LST list the trademarks and the attributes needed for the TM tag. Use the Epic or Frame2000 editor to insert these tags and attributes. For more information about how to mark trademarks, see the *ID Workbench Getting Started and User's Guide*.

Using document classes with XHTML style sheets

This topic is sort of related to highlighting, so we placed it here.

Cascading style sheets (CSS files) are powerful feature of XHTML.[4] CSS files allow you to define the style used for the XHTML outside of your XHTML content itself. This separation of style and content allows you as a writer to concentrate on the content; without always being concerned about the style. You can make changes in the CSS file; without affecting the content of hundreds (or thousands) of web pages.

The ID Workbench XHTML output transform includes a way of passing style elements through to the XHTML. The IBMIDDoc ClassDef tag defines an output class. The Class attribute in a tag indirectly specifies the style for that tag. The setting in the document is reflected as a style sheet setting.

Here's a simple example. Follow the steps here to see how this works for you:

1. We want some chapter headings to look a certain way: 18pt bold. Here's a simple CSS file named MYSTYLE.CSS that has that setting; the style is named "fred".

   ```
   h1.fred { font-size: 18pt; font-weight: bold }
   ```

2. To have this style used in our XHTML; we need to first define a ClassDef tag in our document prolog, in a PropDesc section; like this:

   ```
   <classdef classname="fredthing" eletypes="d" outputclass="fred">
   ```

   You can make the classname and the output class the same; we just wanted to show you that they can be different. The classname specifies the class for an SGML tag; the outputclass specifies the class style in the CSS file.

3. To have a heading have that style; you use the class attribute on the D tag as follows:

   ```
   <d class="fredthing">
   <dprolog>
   <titleblk><title>Flintstone</title></titleblk>...
   ```

4. When you format using ID Workbench; and specify the MYSTYLE.CSS style sheet on the XHTML-3 processing options page; your output will contain a link to your style sheet; and the heading will have a class to point at that style sheet:

   ```
   <h1 class="fred">Flintstone
   ```

4. We won't describe cascading style sheets (CSS files) here; there is plenty of discussion of them on the web.

Chapter 5. Examples, figures, artwork, and multimedia

In IBMIDDoc, all non-text objects are considered multimedia objects, including graphics. This chapter explains how to use these objects in IBMIDDoc. The elements discussed in this chapter include:

- Lines (see "Just plain lines")
- Xmp (see "Examples of computer output" on page 54)
- LitData (see "Literal text data" on page 54)
- MMObj (see "Including artwork in documents" on page 55)
- Fig (see "Figures" on page 57)
- CGraphic (see "Character graphics" on page 59)
- Screen (see "Screens" on page 59).

Just plain lines

Use the Lines element to contain text that has record ends, or boundaries, that need to be preserved when presented to the document user.

The LINES element allows you to control where lines break. That is, within the content of the LINES element, IBMIDDoc will end each output line at the same point where you ended the input line in the markup. In SGML terms, the record ends must be respected. This occurs whether you include text characters within the Lines element, or if you reference an entity that contains text characters or a graphic.

You can include information within LINES by either of the following:

- The Lines tag contains text characters that should be presented "as-is".
- Using the OBJ attribute to name an entity containing the text or other character data to be presented

The next example illustrates using a Lines element to contain unflowed text:

a partridge in a pear tree
two turtledoves
three French hens
four calling birds
five golden rings
six geese a-laying
seven swans a-swimming
eight maids a-milking
nine ladies dancing
ten lords a-leaping
eleven pipers piping
twelve drummers drumming

Here's its markup:

```
<LINES>
a partridge in a pear tree
two turtledoves
three French hens
four calling birds
```

```
five golden rings
six geese a-laying
seven swans a-swimming
eight maids a-milking
nine ladies dancing
ten lords a-leaping
eleven pipers piping
twelve drummers drumming
</LINES>
```

This next example shows how you would code the Lines tag to use the OBJ attribute to insert a file named "lines.txt":

```
<lines obj="samplelines">
```

Here's the declaration:

```
<!ENTITY linestext SYSTEM "lines.txt" ndata linespec>
```

The lines typically have a space before and after them, except at the start of a page, column, or table entry.

Examples of computer output

The Xmp element typically contains an example of computer input or output. Sometimes you may have a very long example — possibly running for several pages. In this case, you have to tell IBMIDDoc that it is okay to break the example at any point after a specified number of lines have printed. You do this with the KEEP attribute. This is illustrated in the example that follows.

```
<XMP STYLE='BKM:(KEEP="10")'>
10 LET A = B
20 IF A GT C THEN GO 40
30 LET A = C
40 PRINT A, C
</XMP>
```

| You can use the LINELENGTH attribute to automatically scale down the example
| to fit the line length you specify. You can use the PGWIDE attribute to make your
| examples page-wide (pgwide=1). The default is pgwide=2; the example is as wide
| as the current textline.

As with Lines, you can also use the OBJ attribute to include the example. In the example that follows, the OBJ attribute is the entity name `sampcprg`. This entity is a small sample C program named `sampcprg.c`. The XMPs element uses the content of this entity as its content.

```
   ⋮
<!ENTITY  sampcprog SYSTEM "sampcprg.c" NDATA C>
   ⋮
<XMP OBJ="sampcprg">
```

The example typically has a space before and after, except at the start of a page, column, or table entry.

Literal text data

Literal data can be used to contain special information or code in which SGML markup is not recognized. It may also refer to the content of other files whose content will not be processed as SGML markup. Examples of this type of data include character translation, special code pages, and samples of programming code.

For example, a sample C++ program could be contained in the LitData element:

```
<xmp>
<LITDATA>
// testprog.cpp
#include <iostream.h>
int main(void)
{
 ...
}
</LITDATA>
</xmp>
```

This same program could be referenced using the OBJ attribute that refers to an entity that contains the program, as shown in the next example.

```
<!ENTITY  testprg SYSTEM "testprg.c" ndata c>
   ⋮
<FIG>
<CAP>A basic C++ program.
<LITDATA OBJ="TESTPRG">
</FIG>
```

Including artwork in documents

In most cases, the artwork you include will be constructed using a graphic/image tool such as CorelDraw or Photoshop. It will be merged with the text during processing for the output device. All you have to do is specify the file type of the artwork that you want to include in your document; the formatter does the rest.

You use the MMObj and ObjRef elements to contain artwork such as images, vector graphics, encapsulated PostScript, or video clips. The processing and presentation systems in use determine the types of multimedia objects that are supported.

In order to use graphic objects in your SGML markup, you must declare them as entities using an entity declaration. Use the notation "graphics" on the declaration.

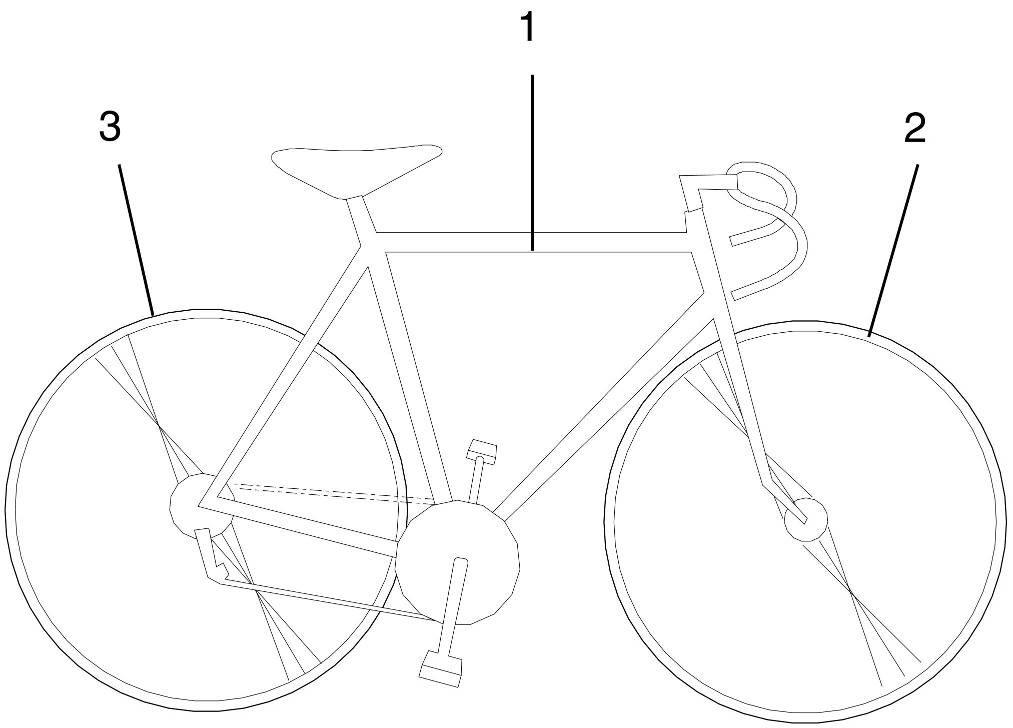
```
<!ENTITY bike system "bike.gif" ndata graphics>
```

In some cases, certain graphic formats are supported for in-line viewing during your editor session. Use the file extension GIF, JPG, TIF, or EPS (if your EPS file has a TIFF header) when you declare your graphic, and the editor will display the artwork.

In the example that follows, the entity is defined first, and it is referred to later in the document by the OBJ attribute on the ObjRef element contained in the MMObj element. Here's the declaration and the markup for an illustration of a bike:

```
<!ENTITY bike system "bike.gif" ndata graphics>
  ...
<MMOBJ>
<OBJREF OBJ="bike">
<TEXTALT>This is a two-wheeled bicycle.</TEXTALT>
</MMOBJ>
```

And here’s the bike:



The MMObj element also contains a TextAlt element. The TextAlt element contains a text description that is presented as an alternative to the artwork.[5] The text appears in HTML when the user’s mouse hovers over the artwork. So be sure to type something meaningful in the TextAlt; don’t be embarrassed by entering something like “Fred, is this really true?” and then having that go out the door and on to a website.

Normally a single space preceeds the artwork. You can use the setting `placement=runin` to have the artwork put inline in a sentence. You can also use `placement=runin` to have the artwork appear in the margin.

Creating graphic links

To create a graphic link for RTF, IPF, or HTML, you need to use the MMObjLnk tag. The Linkend attribute specifies the link ID. The AreaDef element is not used (that is, you cannot create hotspots with it; the entire graphic becomes a link).

Here is an example that will make the graphic part1 link to the ID “newdiv”:

```
<mmobj><objref obj="part1">
<mmobjlink linkend="newdiv">
<areadef coords="1 100">
</areadef></mmobjlink>
<textalt></textalt>
</mmobj>
```

5. Some say a picture is worth a thousand words; well, this is where you put the thousand words to express your picture to those that cannot view pictures.

Figures

Figures typically contain examples, text, or artwork; so it makes sense to talk about them here.

You can choose to have the figure formatted within the column or formatted the full width of the page. You can choose to have it set off with rules across the page, put in a box, or formatted with no frame at all. You can give it an identifier, which will allow you to make cross references to it (which we'll cover in a later chapter). You can give the figure a caption, which will also cause it to be listed in the figure list, if you have one. You can extend the figure caption with a figure description, too (only the caption itself goes in the figure list). And, if you have a very large figure, you can have it split into pieces with a caption that says "Part x of y" on each piece.

Here's a sample, column-wide figure:

Here are some lines
in the sample, simple figure.

Here's its markup:

```
<fig pgwide="0">
<lines>Here are some lines
in the sample, simple figure.</lines>
</fig>
```

The pgwide attribute sets the figure as column-wide.

Your figure can also have a caption or description.

To have a full-page figure, you specify this PgWide attribute:

```
<fig pgwide="1">
<lines>Here are some lines in a page-wide figure.</lines>
</fig>
```

You can choose to add a box around your figure, or to have lines or rules appear before and after your figure. You add these with the FRAME attribute. For example:

```
<fig frame="rules">
```

Or:

```
<fig frame="box">
```

Figure captions and descriptions

Your figures can have a short caption and an optional longer description. The caption can appear in a figure list at the beginning of your book. A good place for the figure list is after your table of contents. If your figure has a caption, you should also give it an ID; some processes (like BookMaster) complain about figures that have captions but are missing the ID. Put the ID on the fig tag, not on the caption. The caption should be entered like a heading, without ending punctuation.

Here's another sample figure. This one is page-wide and includes a caption:

Here are some lines
in the sample, simple figure.

Figure 4. Here's a sample, page-wide figure

Here's its markup; note the FIG tag has an identifier:

```
<fig id="samplefig" style="bkm:(place=inline width=page)">
<cap>Here's a sample, page-wide figure</cap>
<lines>Here are some lines
in the sample, simple figure.</lines>
</fig>
```

Here's another sample figure with both a caption and a description. You enter a description like a sentence, with punctuation. Note that the caption source has no punctuation; if any is needed, it is added by the formatter.

Here are some lines
in the sample, simple figure.

Figure 5. Here's a sample figure with a caption and description. This figure has a description. Note that descriptions have punctuations like sentences.

Here's its markup:

```
<fig id="samplefigdesc" style="bkm:(place=inline width=column)">
<cap>Here's a sample figure with a caption and description
</cap>
<desc>This figure has a description. Note that descriptions
have punctuations like sentences.</desc>
<lines>Here are some lines
in the sample, simple figure.</lines>
</fig>
```

Multipart figures

There are times when your figure is too big to fit on a single page. Sorry, there's no way to automatically split a figure. The FIGSEG (figure segment) tag is used to specify the points at which the figure can be broken into parts. The formatter takes care of printing the figure caption with the "Part x of y" on each part for you. Figure 6 shows a sample two-part figure.

In Xyvision PostScript output, the figures are split at the FIGSEG tag. For HTML, the figure segments are ignored; the output appears as if it were one figure.

First part of the figure.
Figure 6. My little caption (Part 1 of 2). My little description.

Last part of the figure.
Figure 6. My little caption (Part 2 of 2). My little description.

Here's its source:

```
<fig id="idgifsegxmp"><cap>My little caption</cap>
<desc>My little description.</desc>
<figseg>
<p>First part of the figure.</p>
</figseg>
<figseg>
<p>Last part of the figure.</p>
</figseg>
</fig>
```

Character graphics

Use the CGraphic element to contain a graphic created using character graphics, such as box and line characters.

Note: Ensure the cgraphics are external file entities. Having them inline increases the chances of data corruption.

CGraphic will often use the OBJ attribute to reference an entity that contains the actual cgraphic markup characters. The NOTATION attribute value is always LINESPEC.

```
<!ENTITY ILLUS01 SYSTEM "ILLS01.CHG" ndata linespec>
   ⋮
<CGRAPHIC OBJ="ILLUS01">
```

You can use the LINELENGTH attribute to automatically scale down the character graphic to fit the line length you specify.

Screens

Use the Screen element to contain or refer to a representation of a computer display or panel. There are several different ways to use this element in IBMIDDoc.

The Screen element is designed to hold “green” screen images which use fixed pitch fonts. The Screen element can reference an entity using the OBJ attribute.

```
<SCREEN OBJ="scr1">
```

It can also just contain the content:

```
<screen>
 EDFUSCRN  SCRIPT   A1  V 132  TRUNC=132 SIZE=69 LINE=45 COL=1 ALT=4

=====
=====
=====
=====
=====       This is what
=====
=====           the screen looked like
 ...
</screen>
```

The resulting output from a typical screen display would look similar to the display illustrated below.

```
 EDFUSCRN  SCRIPT   A1  V 132  TRUNC=132 SIZE=69 LINE=45 COL=1 ALT=4

=====
=====
=====
=====
=====        This is what
=====
=====            the screen looked like
=====
=====                when I was writing
=====
=====                    this section on screens.
=====
=====         ALL LINES IN THE SCREEN MUST BE ACCOUNTED FOR
=====
=====                    IN THE SOURCE FILE
=====
=====
=====
=====

====>
                                                              XEDIT  1 FILE
```

You can use the LINELENGTH attribute to automatically scale down the screen to fit the line length you specify. You can use the PGWIDE attribute to make your screens page-wide (pgwide=1). The default is pgwide=2; the screen is as wide as the current textline.

Math formulas

Mathematical formulas are contained by the Formula element. Only formulas created using the Script Mathematical Formula Formatter (SMFF) are supported at this time. This is only supported for BookMaster hardcopy output.

```
<FORMULA NOTATION="smff">
integral from 0 to infinity of d x
</FORMULA>
```

Chapter 6. Cross-referencing

At last we're going to tell you what all those ID attributes are for!

IBMIDDoc manages cross references to headings, figures, tables, items in ordered lists, and footnotes, indeed, to any spot in a document. Each of these types of cross references is done with the ID (identification) and REFID (reference identification) attributes — ID on the thing pointed to, and REFID on the thing doing the pointing.

Also, when you take your BookMaster source and create an online document, IBMIDDoc uses the REFID and ID attributes to set up hypertext links between cross references within and between documents. See Chapter 12, "All about linking" on page 129 for more information about interdocument cross-reference linking.

The ID (identification) attribute is a way of giving something a name that IBMIDDoc can use in providing a reference to that thing. You've already seen cases in this book of cross references to headings, although you may not have realized it at the time. For example, this is a cross reference to a heading:

Formatted Example

See Chapter 3, "All kinds of lists" on page 29.

End of Formatted Example

To get the cross reference, two things have to be done. First of all, the heading that we referred to was entered like this:

```
<d id="lsts">
<dprolog><titleblk>
<title>All kinds of lists</title>
</titleblk></dprolog>
```

The "lsts" is a name we made up. It can be any combination of letters and numbers as long as it is no more than 64 letters and numbers. The first character must be a letter; and the other characters can be a period (.) or a dash (-). You can use uppercase or lowercase letters. Each ID in the document must be unique. The other thing you have to do is enter the cross reference itself, using the XREF tag with a REFID attribute. It would look like this:

```
See <xref refid="lsts">.
```

Because we used the same value for the REFID attribute as for the ID attribute of the heading we are referring to, IBMIDDoc knows which one we want. Because it also knows what page it is on, it supplies that, too. If the heading had been on the same page as the reference, IBMIDDoc would know that also, and it would not give the page number.

The target of a cross-reference should be to an ID on the outer container (for example, D, MSG, LE, FIG, TABLE) and not the title text or caption text.

Since online BookManager® books don't have actual pages, cross references will refer to topic numbers. For online HTML and other information, the references just contains the heading.

Cross references can go in either direction; that is, the thing being referred to can come before or after the cross reference.

It is a good idea to pick descriptive names for your ID attributes; if you name things "chap1", "chap2", and so on, you will find that when you update the source file and insert, delete, and rearrange material, your names will be more confusing than useful in trying to keep track of what is going on.

There are times when you want to control the form of cross references using the FORM attribute, as described in "Controlling the form of cross references" on page 65.

You can use XREF to reference any heading level and to reference anything. These next topics show cross references to common elements:

- "Referencing a figure"
- "Referencing a table" on page 63
- "Referencing a list item" on page 63
- "Referencing anything at all" on page 64

Referencing a figure

To reference a figure, ensure the figure has a caption. Here's a sample figure that we're going to reference in a bit:

Formatted Example

A figure that is going to have a cross reference must also have a caption.
Figure 7. Captioned figure for cross reference

End of Formatted Example

Here's the markup for our little figure above:

```
<fig id="littlefig">
<cap>Captioned figure for cross reference</cap>
<p>A figure that is going to have a cross reference
must also have a caption.</p>
</fig>
```

Now we can code and XREF like this:

```
See <xref refid="littlefig"> for a sample of a figure
reference.
```

and the result is this:

Formatted Example

See Figure 7 for a sample of a figure reference.

End of Formatted Example

Referencing a table

Cross referencing a table is just like referencing a figure. The table must have a caption. Here's a sample table:

Formatted Example

Table 2. Captioned table for cross reference

A table that is going to have a cross reference...	
	... must also have a caption.

End of Formatted Example

Here's the markup for our little table above:

```
<table pgwide="0" id="sampletable">
<cap>Captioned table for cross reference</cap>
<tgroup cols="2">
<colspec colname="col1">
<colspec colname="col2">
<tbody><row>
<entry colname="col1">A table that is going to have
a cross reference...</entry>
<entry colname="col2"></entry>
</row><row>
<entry colname="col1"></entry>
<entry colname="col2">... must also have a caption.
</entry></row></tbody></tgroup></table>
```

Now we can code and XREF like this:

```
See <xref refid="sampletable"> for a sample of a table
reference.
```

and the result is this:

Formatted Example

See Table 2 for a sample of a table reference.

End of Formatted Example

Referencing a list item

Another common phenomenon in the books we produce is the cross reference to an item in an ordered list. To do this, we put an ID attribute on the LI (list item) tag for the item we want to point to, and use an XREF tag with a REFID attribute to do the pointing. Here's a sample of some new tax instructions from the U.S Department of Treasury:

Formatted Example

1. If the amount on line 37 is greater than the lesser of lines 5 and 6, go to step 5 on page 64.
2. Enter the sum of line 5 and line 37. If this exceeds your total annual income before deductions, go to step 6 on page 64.

3. If line 32 less the difference of lines 73 and 74 on Schedule C is greater than line 36 plus line 17 of Schedule A and you are under 65 years of age, go to step 1 on page 63, where you will be in a loop until you are 65.
4. Use table 30-C to compute the number of bathrooms in your house and enter on line 56.
5. Enter the root-mean-square of line 14.
6. Sign form and mail with remittance.

End of Formatted Example

Here's its markup:

```
<ol>
<li id="ageloop">If the amount on line 37 is greater
than the lesser of lines 5 and 6, go to
step <xref refid="squareit">.</li>
<li>Enter the sum of line 5 and line 37. If this exceeds
your total annual income before deductions, go to
step <xref refid="mailit">.</li>
<li>If line 32 less the difference of lines 73 and
74 on Schedule C is greater than line 36 plus line
17 of Schedule A and you are under 65 years of age,
go to step <xref refid="ageloop">, where you will
be in a loop until you are 65.</li>
<li>Use table 30-C to compute the number of bathrooms
in your house and enter on line 56.</li>
<li id="squareit">Enter the root-mean-square of line
14.</li>
<li id="mailit">Sign form and mail with remittance.
</li>
</ol>
```

Referencing anything at all

Although we've given you a lot of ways to create cross references, there are times when none of those ways exactly meets your requirements. So IBMIDDoc has IDs on all its tags, which allow you to identify any spot in your document, by page number, and to refer to it from another place.

A tag with an ID attribute has no effect on the formatting of the text around it, but to get the result you want, it should be placed in the same places that are good for index entries. These are described in "Where to put index entries" on page 119. You can also specify some text that is associated with this tag, using the XREFTEXT (cross-reference text) attribute on that tag. If there was no XREFTEXT, then the XREF gives you just the page number of the tag. If there was XREFTEXT, then XREF prints that text, followed by "on page" and the page number.

For example, here's a paragraph we want to reference:

Here's my little paragraph that I want to reference.

Here's its markup, including xreftext:

```
<p id="samplepara" xreftext="My litle paragraph">
Here's my little paragraph that I want to reference.
</p>
```

Then, to reference that, we would code the following:

```
See <xref refid="samplepara"> for a small bit of information.
```

And we would get this result:

See “My litle paragraph” on page 64 for a small bit of information.

Controlling the form of cross references

The “normal” form of a cross reference, as shown in the examples in the preceding sections, is not always exactly suitable to your needs. IBMIDDoc has a FORM attribute on the XREF tag that allows you to control what is generated. Table 3 shows the supported values of the FORM attribute for hardcopy output, for the typical things you cross-reference.

Table 3. XREF forms for hardcopy output

Referenced item	**Form=**				
	Normal[1]	**Full**	**Text**	**Location**	**Number**
Heading	Chapter 6, “Cross-referencing” on page 61	Chapter 6, “Cross-referencing” on page 61	Chapter 6, “Cross-referencing”	61	Chapter 6
Figure	Figure 7 on page 62	Figure 7 on page 62	Figure 7	62	7
Table	Table 2 on page 63	Table 2 on page 63	Table 2	63	2
List item	1 on page 63	1 on page 63	1	63	1
Anything (first paragraph in this chapter with xreftext)	“What are the IDs for?” on page 61	“What are the IDs for?” on page 61	What are the IDs for?	61	

[1] The “on page *pagenumber*” only appears if the referenced item is on a different page.

Note that punctuation after a quoted reference remains outside the quote. It does not “move inside” the quote as the IBM style guidelines recommend. This is currently a Xyvision formatter and Frame2000 formatter restriction; it may be removed in the future.

Chapter 7. Creating IBMIDDoc Tables

IBMIDDoc supports the CALS table elements, with a few modifications for IBM-specific usage. This section describes fundamental IBMIDDoc table concepts, and gives examples of markup for several tables.

Migration Note

For BookMaster tables that include DVCF text that will be reused across a number of documents, or will be output in many different formats, use the IBMIDDoc modular information elements instead of table elements.

IBMIDDoc Table Markup Concepts

You can use IBMIDDoc to create a wide variety of tables. From simple tables:

Table 4. Simple table

A	**B**	**C**
D	E	F

To complex tables:

Table 5. Complex table

A	**B**	**C**
	D	**E**
F	G	

You can select several different combinations of rules and framing. You can have a table heading and a table caption. You can specify the alignment and formatting of text in your table. You can combine several table definitions within a single table. But you don't have to use all of the power of table markup every time you want to create a table. You can get handsome tables with fairly simple markup. So we'll take things one step at a time, beginning with simple table markup and moving on to the more advanced stuff later.

IBMIDDoc tables are contained in a Table element. The Table element then can contain TGroup elements. In most cases, you'll assign values to the TGroup's attributes that define the structure and layout of the table.

Creating simple tables

Tables consist of **cells**, arranged in **rows**. Here is a simple example:

Table 6. A simple example

Header 1	**Header 2**	**Header 3**
Row 1, Cell 1	Row 1, Cell 2	Row 1, Cell 3; here's a little more text than the other cells have

Table 6. A simple example (continued)

Header 1	Header 2	Header 3
Row 2, Cell 1	Row 2, Cell 2	Row 2, Cell 3
Table footer		

Each little box in this table is a cell. This table consists of two rows, with three cells in each. You can, of course, have more or fewer rows and cells, so this basic form can take care of a lot of your table requirements. A table must have at least one row (in addition to any headings, footings, and captions) to produce any output, and each row must have at least one cell.

The markup for this table is as follows:

```
<table><cap>A simple example</cap>
<tgroup cols="3">
<colspec colname="col1">
<colspec colname="col2">
<colspec colname="col3">
<thead>
<row>
<entry colname="col1" valign="top">Header 1</entry>
<entry colname="col2" valign="top">Header 2</entry>
<entry colname="col3" valign="top">Header 3</entry>
</row>
</thead>
<tfoot>
<row>
<entry namest="col1" nameend="col3" valign="top" align="center">
Table footer</entry>
</row>
</tfoot>
<tbody>
<row>
<entry colname="col1">Row 1, Cell 1</entry>
<entry colname="col2">Row 1, Cell 2</entry>
<entry colname="col3">Row 1, Cell 3; here's a little
more text than the other cells have</entry>
</row>
<row>
<entry colname="col1">Row 2, Cell 1</entry>
<entry colname="col2">Row 2, Cell 2</entry>
<entry colname="col3">Row 2, Cell 3</entry>
</row>
</tbody>
</tgroup>
</table>
```

We began the table with the TABLE tag and put in a caption with CAP. Next comes TGROUP with the number of columns in the table, and COLSPECs indicating the column names. The table header, THEAD, contains the column headings. Some tables have table footers, contained in the TFOOT element. The table content starts with the TBODY tag. We then started specifying rows and cells with the ROW and ENTRY tags. The text within each cell is formatted the same way regular body text is formatted (we'll look later at how to redefine this.) Finally, we ended the table with the proper ending tags.

Table headers repeat at the top of each table part for multi-part tables. The Xyvision formatter places a table footer once, at the end of the table, for each part of a multi-part figure. Frame2000 currently always repeats the table foot at the bottom of each page of a multi-part table.

Specifying table column widths

If you want your columns to have different widths, like this:

Table 7. Simple table with different column widths

a	b	c

You use the COLSPEC tag and the COLWIDTH attribute, like this:

```
<table><cap>Simple table with different column widths
</cap>
<tgroup cols="3">
<colspec colnum="1" colname="col1" colwidth="1*">
<colspec colnum="2" colname="col2" colwidth="1*">
<colspec colnum="3" colname="col3" colwidth="2*">
<tbody>
<row>
<entry colname="col1">a</entry>
<entry colname="col2">b</entry>
<entry colname="col3">c</entry>
</row>
</tbody>
</tgroup>
</table>
```

In simple table markup, cell widths are the same as the corresponding column widths, so COLWIDTH really specifies the cell widths. Omitting the COLWIDTH from all the COLSPECS causes each column to have the same width. In our example, the Xyvision formatter makes the whole table as wide as the text column (this is the initial setting) and calculates the table columns based on that.

Of course, you won't always want all your table columns to be the same width. The COLWIDTH attribute and the asterisks tell the formatter that we want the table divided porportionally; we don't care what the exact width turns out to be. So the formatter decides for us. You can specify 1*, 1*, and 2* if you want the first and second columns to have the same width, and to have last column be twice as wide as the first or second column.

| When text is too big for a table cell, the Xyvision formatter continues to flow the
| text into the next column or off the edge of the page; it currently issues no
| message. You should check your output to ensure formatting is as you desire.

Table captions and descriptions

Your tables can have a short caption and an optional longer description. The caption can appear in a table list at the beginning of your book. A good place for the table list is after your table of contents. If your table has a caption, you should also give it an ID; some processes (like BookMaster) complain about tables that have captions but are missing the ID. Put the ID on the table tag, not on the caption. The caption should be entered like a heading, without ending punctuation.

Here's a sample table with a small caption:

Table 8. Sample table caption

my little
sample table

Here's its markup:

```
<table pgwide="0" id="tablesample">
<cap>Sample table caption</cap>
<tgroup cols="1">
<colspec colname="col1">
<tbody>
<row>
<entry colname="col1">my little</entry>
</row>
<row>
<entry colname="col1">sample table</entry>
</row>
</tbody>
</tgroup>
</table>
```

Here's another sample table with both a caption and a description. You enter a description like a sentence, with punctuation. Note that the caption source has no punctuation; if any is needed, it is added by the formatter.

Table 9. Sample table caption. This table shows little remarkable information. You need to read between the lines.

my little
sample table

Here's its markup:

```
<table pgwide="0" id="tablesampledesc">
<cap>Sample table caption</cap>
<desc>This table shows little remarkable information.
You need to read between the lines.</desc>
<tgroup cols="1">
<colspec colname="col1">
<tbody>
<row>
<entry colname="col1">my little</entry>
</row>
<row>
<entry colname="col1">sample table</entry>
</row>
</tbody>
</tgroup>
</table>
```

If you want a table to have a description, but no caption, you can do that too:

Fred's little table that he doesn't want numbered

Here's something Fred likes to talk about.	
	The secret to good ice fishing is keeping your worms warm.

Here's its markup:

```
<table><desc>Fred's little table that he doesn't want
numbered</desc>
<tgroup cols="2">
<colspec colname="col1">
<colspec colname="col2">
<tbody>
<row>
<entry colname="col1">Here's something Fred likes
```

```
to talk about.</entry>
<entry colname="col2"></entry>
</row>
<row>
<entry colname="col1"></entry>
<entry colname="col2">The secret to good ice fishing
is keeping your worms warm.</entry>
</row>
</tbody>
</tgroup>
</table>
```

Page, column, and line-wide tables

You use the PGWIDE attribute on the TABLE tag to control the width of a table.

PGWIDE=0
(zero) makes the table column-wide

pgwide=0	

PGWIDE=1
(one) makes the table page-wide.

pgwide=1	

PGWIDE=2
(two) makes the table as wide as the current text line.

pgwide=2	

In BookMaster, tables default to page wide. In Xyvision, tables default to column wide.

If you need to have wide tables in BookManager BOOKs, use the DWIDTH attribute to specify a wide display width. The normal setting is 75:

```
style=bkm:(dwidth=100)
```

Complex tables are NOT supported in RTF nor IPF. Complex tables are tables whose contents include lists, definition lists, etc. or tables with complex column/row spanning. If you are creating RTF or IPF output, you should avoid making use of the complex features that the Epic table editor enables for you.

Splitting tables between pages

In BookMaster hardcopy, tables do not split. This may cause a BookMaster error and the second part of the table will not have a caption. Use this override on your large tables so they split in BookMaster without errors:

```
bkm:(split=yes)
```

In Xyvision, tables always split. You should accept this default due to possible translation center impacts (expansion space in rows). If you can't bear to split your table, you can use this override:

```
bkm:(split=no)
```

If you have a table row that contains a **lot** of information, more information than will fit on a hardcopy page for your document's style, you will get an error during Xyvision PostScript formatting. You will need to split the large row into two or more smaller rows.

Affecting how a table appears: Rules, Separators, Shading

The Table tag allows several attributes and settings to control how your table appears. These are all available in the Xyvision PostScript formatter. The other transforms support some of these settings, but not all of them. Experiment with your desired output formats to determine what you can use.

- To make the table have a frame, rules, or nothing, the FRAME attribute can be set to:

  all Rules appear on all four sides; the table is boxed.

frame=all	

  bottom

frame=bottom	

  none

frame=none	

  sides

frame=sides	

  top

frame=top	

  topbot

frame=topbot	

- To control the vertical rules in the table, use the COLSEP attribute.

  0 (zero) makes the table have no column separators

colsep=0	
	No vertical rules appear

  1 (one) makes the table have column separators

colsep=1	

	The vertical rules appear

- To control the horizontal rules in the table, use the ROWSEP attribute.

  0 (zero) makes the table have no row separators

rowsep=0	No horizontal rules appear

  1 (one) makes the table have row separators

rowsep=1	
	The horizontal rules appear

- To shade an entire table, a row, or a cell, use the SHADE attribute:

  Shade=NOShade | XLight | Light | Meduim | Dark | XDark
  Use the Shade attribute to specify the shading. This table example shows the shading values used in table cells:

Table 10. Cell entry shading. In the editor, use the modify attributes icon or Ctrl-A to set the Shade attribute for the cell's Entry tag.

noshade (0%)	xlight (5%)	light (26%)	medium (50%)	dark (74%)	xdark (100%)
the	quick brown	fox	jumps over	the lazy	

- To rotate a table, set the ORIENT attribute to LAND.

  port Use this for normal, portrait-oriented tables.

orient=port	

  land Use this to turn your table on it's side (landscape) for hardcopy. PGWIDE is ignored (it is assumed to be full-page).

orient=land	

Defining the Column Specifications

A typical set of ColSpec attributes for a simple table is shown in the example that follows.

Migration Note

At this time, the graphical table editor does not handle a mixture of proportional and fixed COLWIDTH values in the same table. You should use proportional COLWIDTHs until this issue is resolved.

```
<TABLE FRAME="ALL">
 <TGROUP COLS="4" COLSEP="1" ROWSEP="1" ORIENT="PORT" PGWIDE="0">
  <COLSPEC COLWIDTH="68*">
  <COLSPEC COLWIDTH="127*">
  <COLSPEC COLWIDTH="195*">
  <COLSPEC COLWIDTH="66*">

   ⋮
```

The ColSpec attributes include:

COLNUM=*col_number*
This value indicates the number of the column.

COLNAME=*col_name*
Specifies the column name. This name is referenced by other table elements.

ALIGN=LEFT | RIGHT | CENTER | JUSTIFY | CHAR
This attribute specifies the horizontal positioning of text in the column.

- **LEFT**
  specifies left alignment (the default).
- **RIGHT**
  specifies right alignment.
- **CENTER**
  specifies center alignment.
- **JUSTIFY**
  specifies that the text should be justified.
- **CHAR**
  specifies the character that is used for alignment.

CHAROFF=*number*
Specifies the character offset for Entry elements in a column.

COLWIDTH=*measure*
Specifies a fixed, proportional, or mixed measure for the column width.

Migration Note

At this time, mixed measures are not supported. You should use proportional measures.

COLSEP=0 | 1
This attribute's value specifies that the internal column rules should be:
- to the right of each cell's content (1)
- not displayed at all (0)

Note that BookMaster does not allow rules on only one side of a column.

ROWSEP 0 | 1
This attribute's value specifies that the internal row rules should be:

- below each Entry element that ends a row (1)
- not displayed at all (0)

Note that BookMaster does not allow rules on only one side of a row.

Defining Rows and Entrys

Rows are defined using the Row element. Each Entry element contained in a Row element occupies the consecutive column, from left to right. The two Row attributes you will use are VALIGN (vertical alignment) ALIGN (horizontal alignment). Unless the ROWSEP attribute is specified, the Row inherits the ROWSEP value specified on the Table or TGroup element.

```
<TABLE FRAME="ALL">
 <TGROUP COLS="3" COLSEP="1" ROWSEP="1">
  <COLSPEC COLWIDTH="152*">
  <COLSPEC COLWIDTH="152*">
  <COLSPEC COLWIDTH="152*">
   <TBODY>
    <ROW>
     <ENTRY VALIGN="TOP" ALIGN="LEFT">ROW 1, CELL 1</ENTRY>
     <ENTRY VALIGN="TOP" ALIGN="LEFT">ROW 1, CELL 2</ENTRY>
     <ENTRY VALIGN="TOP" ALIGN="LEFT">ROW 1, CELL 3;
     HERE'S A LITTLE MORE TEXT THAN THE OTHER CELLS HAVE.</ENTRY>
   </ROW>

   ⋮
```

To have unformatted text in a table, insert the LINES tag, then insert your cell content within the lines.

Making your tables accessible

The requirement to make your documents accessible to screen readers is very important. Tables are one of the more challenging items. IBMIDDoc has items to help your tables be accessible.

First, ensure you have marked your table headers as a header row, so that the THEAD tags are used. Ensure you are not just using a ROW tag and have made the headings bold with a style. This allows the XHTML, when read by a screen reader, to indicate your proper intention to the reader.

Second, if your table uses the first column as a row header; indicate that by using the RowHeader attribute:

RowHeader=FirstCol | <u>NoRowHeader</u>
This specifies whether the first column is a row header. If your table's first column is really a row-header, specify the RowHeader=FirstCol setting. In the same way that a column header introduces a table column; the row header introduces the table row. This is to help make tables, whose first column is a row-header, to be more accessible when the output is for XHTML. The default is NoRowHeader. Here's an example of a table where the FirstCol attribute should be used:

Switch Location	**Setting**
Hallway	On

Switch Location	**Setting**
Kitchen	Off
Bedroom	On

And the markup:

```
<table pgwide="2" rowheader="firstcol">
<tgroup cols="2">
<colspec colname="col1">
<colspec colname="col2">
<thead>
<row>
<entry colname="col1" valign="top">Switch Setting
</entry>
<entry colname="col2" valign="top">Value</entry>
</row>
</thead>
<tbody>
<row>
<entry colname="col1">Hall switch</entry>
<entry colname="col2">On</entry>
</row>
<row>
<entry colname="col1">Kitchen switch</entry>
<entry colname="col2">Off</entry>
</row>
<row>
<entry colname="col1">Bedroom switch</entry>
<entry colname="col2">On</entry>
</row>
</tbody>
</tgroup>
</table>
```

A Few Simple Table Examples

Let’s look at a couple of simple tables. This section includes the IBMIDDoc markup, and an approximation of the resulting formatted output.

A Simple Table

Let’s look at a simple IBMIDDoc table

Table 11. A simple example

Row 1, Cell 1	Row 1, Cell 2	Row 1, Cell 3; here’s a little more text than the other cells have
Row 2, Cell 1	Row 2, Cell 2	Row 2, Cell 3

Here’s its markup:

```
<table frame="all" pgwide="0">
<cap>A simple example</cap>
<tgroup cols="3" colsep="1" rowsep="1">
<colspec colname="col1" colwidth="1*">
<colspec colname="col2" colwidth="1*">
<colspec colname="col3" colwidth="1*">
<tbody>
<row>
<entry valign="top">Row 1, Cell 1</entry>
<entry valign="top">Row 1, Cell 2</entry>
<entry valign="top">Row 1, Cell 3; here's a little
```

```
more text than the other cells have</entry>
</row>
<row>
<entry valign="top">Row 2, Cell 1</entry>
<entry valign="top">Row 2, Cell 2</entry>
<entry valign="top">Row 2, Cell 3</entry>
</row>
</tbody>
</tgroup>
</table>
```

A Simple Table with More Options

Now let's take a similar table and add another column.

Table 12. Another simple table

Row 1, Cell 1	Row 1, Cell 2	Row 1, Cell 3; here's a little more text than the other cells have	Row 1, Cell 4
Row 2, Cell 1	Row 2, Cell 2	Row 2, Cell 3	Row 2, Cell 4

Here's its markup:

```
<table frame="all" pgwide="0">
<cap>Another simple table</cap>
<tgroup cols="4" colsep="1" rowsep="1" style="BKM:(cols='1* 2* 3* 1*')">
<colspec colname="col1" colwidth="1*">
<colspec colname="col2" colwidth="2*">
<colspec colname="col3" colwidth="3*">
<colspec colname="col4" colwidth="1*">
<tbody>
<row>
<entry valign="top">Row 1, Cell 1</entry>
<entry valign="top">Row 1, Cell 2</entry>
<entry valign="top">Row 1, Cell 3; here's a little
more text than the other cells have</entry>
<entry valign="top">Row 1, Cell 4</entry>
</row>
<row>
<entry valign="top">Row 2, Cell 1</entry>
<entry valign="top">Row 2, Cell 2</entry>
<entry valign="top">Row 2, Cell 3</entry>
<entry valign="top">Row 2, Cell 4</entry>
</row>
</tbody>
</tgroup>
</table>
```

A Simple Table with a Table Header and IBMIDDoc Elements

Now let's take a similar table and add a THead element and some IBMIDDoc Phrase elements with STYLE attribute specifications.

Table 13. Another sample table

Col #1	**Col #2**	**Col #3**	**Col #4**
Row 1, Cell 1	1. Row 1 2. Cell 2	Row 1, Cell 3; here's a little more text than the other cells have	Row 1, Cell 4
Row 2, Cell 1	Row 2, Cell 2	*Row 2, Cell 3*	Row 2, Cell 4

Here's its markup:

```
<table frame="all" pgwide="0">
<cap>Another sample table</cap>
<tgroup cols="4" colsep="1" rowsep="1">
<colspec colname="col1" colwidth="1*">
<colspec colname="col2" colwidth="2*">
<colspec colname="col3" colwidth="3*">
<colspec colname="col4" colwidth="1*">
<thead>
<row>
<entry valign="top" rowsep="1">Col #1</entry>
<entry valign="top" rowsep="1">Col #2</entry>
<entry valign="top" rowsep="1">Col #3</entry>
<entry valign="top" rowsep="1">Col #4</entry>
</row>
</thead>
<tbody>
<row>
<entry valign="top">Row 1, Cell 1</entry>
<entry valign="top"><ol>
<li>Row 1</li>
<li>Cell 2</li>
</ol></entry>
<entry valign="top">Row 1, Cell 3; here's a little
more text than the other cells have</entry>
<entry valign="top">Row 1, Cell 4</entry>
</row>
<row>
<entry valign="top">Row 2, Cell 1</entry>
<entry valign="top">Row 2, Cell 2</entry>
<entry valign="top"><ph style="italic">Row 2, Cell
3</ph></entry>
<entry valign="top">Row 2, Cell 4</entry>
</row>
</tbody>
</tgroup>
</table>
```

A Complex Table with Row and Column Spans

Now let’s take a similar table and add NAMEST, NAMEEND, and MOREROWS attribute values to change the look of the table. NAMEST and NAMEEND specify the spanning of columns, and MOREROWS specifies the spanning of rows.

Table 14. Complex table example

Row 1, Cell 1		Row 1, Cell 2
Row 1, Cell 3	Row 1, Cell 4	

Here’s its markup:

```
<table frame="all" pgwide="0" id="complxt"><cap>Complex table example</cap>
<tgroup cols="3">
<colspec colname="col1" colwidth="77*">
<colspec colname="col2" colwidth="100*">
<colspec colname="col3" colwidth="119*">
<tbody>
<row>
<entry namest="col1" nameend="col2">Row 1, Cell 1
</entry>
<entry colname="col3" morerows="1">Row 1, Cell 2</entry>
</row>
<row>
<entry colname="col1">Row 1, Cell 3</entry>
<entry colname="col2">Row 1, Cell 4</entry>
```

```
</row>
</tbody>
</tgroup>
</table>
```

A Complex Table Header

Here’s a complex table header. As in the previous example, NAMEST, NAMEEND, and MOREROWS were used to combine the heading cells.

Head 1	**Head 2**		
	Sub 1	**Sub 2**	**Sub 3**
a	b	c	d

Here’s it’s markup:

```
<table frame="all" pgwide="0">
<tgroup cols="4">
<colspec colname="col1">
<colspec colname="col2">
<colspec colname="col3">
<colspec colname="col4">
<thead>
<row>
<entry colname="col1" morerows="1" align="center">Head 1</entry>
<entry namest="col2" nameend="col4" align="center">Head 2</entry>
</row>
<row>
<entry colname="col2" align="center">Sub 1</entry>
<entry colname="col3" align="center">Sub 2</entry>
<entry colname="col4" align="center">Sub 3</entry>
</row>
</thead>
<tbody>
<row>
<entry colname="col1">a</entry>
<entry colname="col2">b</entry>
<entry colname="col3">c</entry>
<entry colname="col4">d</entry>
</row>
</tbody>
</tgroup>
</table>
```

Adding footnotes to a table

While you cannot have footnotes within a table using the FN tag, you can use superscripts and a note list to have the same affect. For example, here’s a table with some sample notes:

Sample^1	And another^2
Notes: 1. The first table note 2. And another table note.	

Here is its coding:

```
<table pgwide="0">
<tgroup cols="2">
<colspec colname="col1">
<colspec colname="col2">
```

```
<tbody>
<row>
<entry colname="col1">Sample<ph style="superscript">1</ph></entry>
<entry colname="col2">And another<ph style="superscript">2</ph></entry>
</row>
<row>
<entry namest="col1" nameend="col2">
<notelist>
<li>The first table note</li>
<li>And another table note.</li>
</notelist></entry>
</row>
</tbody>
</tgroup>
</table>
```

Chapter 8. The document structure of an IBMIDDoc document

This section illustrates the order and usage of elements in creating a document. The high-level elements include:

- The IBMIDDoc tag itself ("About the IBMIDDoc tag")
- Prolog ("About the prolog" on page 88)
- FrontM, front matter[6] ("Front matter (FrontM)" on page 98)
- Body (already covered previously, see "Creating the body of your document" on page 20)
- BackM, back matter("About back matter (BackM)" on page 101)

This example gives an high-level view of the structure of an IBMIDDoc document:

```
<ibmiddoc>
<prolog>
  ...
</prolog>
<frontm>
 ...
</frontm>
<body>
<d>
 ...
</d>
</body>
<backm>
 ...
</backm>
</ibmiddoc>
```

About the IBMIDDoc tag

The IBMIDDoc tag contains information about the whole document. This includes the document style, language used, security classification, and page-numbering information.

Getting in style, the document style, that is

IBMIDDoc has a number of built-in styles. To use a style other than the default, set the IBMIDDoc tag's DocStyle attribute to one of these following values:

IBM8X11
8-1/2 by 11 inch style. Replaces BookMaster style IBMXAGD.

IBM7X9
7 by 9 inch style. Replaces BookMaster style IBMXGGD.

IBM2COL
8.5x11 style (2 column layout)

IBMCD
4.75x4.75 style (for CD Jewel Case booklets)

IBMREFC
Reference cards (3-5/8x9in.).

6. Some folks call this "don't matter", because customers seldom read it – do you read a preface?

IBM5X8
5.5x8.5 style (for hardware).

IBM4X6
4.25x6.25 style (for hardware).

IBM8X5
5.5x8.5 landscape style (for hardware).

IBM9X7
7x9 landscape style.

If you create a PDF from this style, the pages may switch between landscape and portrait presentation in Adobe Acrobat Reader or Exchange. Add the following lines to your PostScript file before distilling it to prevent this from occurring:

```
/currentdistillerparams where {pop}
{userdict /currentdistillerparams {1 dict} put} ifelse
/setdistillerparams where {pop}
{userdict /setdistillerparams {pop} put} ifelse
<< /AutoRotatePages /All >> setdistillerparams
```

IBMLAND
Printer System's landscape books. (Not for BookMaster)

IBMXAGD
User Guides (8.5x11in., A4); old BookMaster style.

IBMXARF
Reference (8.5x11in., A4); old BookMaster style. This can be replaced by using a style of ibm8x11 and a layout of onecol.

IBMXGGD
Summary Guides (7-3/8x9in.); old BookMaster style.

TIV7X9
7x9 style for Tivoli®

This style creates automatic running headers for titles. The style puts Part, Chapter, and Tivoli head 1 text in the RETKEY area. The STitle content, if specified, replaces the Title content in the running heading.

TIV8X11
8.5x11 style for Tivoli

This style creates automatic running headers for titles. The style puts Part, Chapter, and Tivoli head 1 text in the RETKEY area. The STitle content, if specified, replaces the Title content in the running heading.

OBIPORT
5.5x8.5 style (for Options by IBM)

OBIWWA6P
4.25x5.75 style (for Options by IBM)

SMALLFLG
3.625x8.5 style (for hardware)

| You can also vary the number of columns in the document. The IBMIDDoc tag has
| a Layout attribute; it has these values:

| **LAYOUT=Default-Layout | OneCol | OffsetCol | TwoCol**
| Specifies the column-style for the book.

Default-Layout
The section uses the default layout for the document style.

OneCol
The headings and text format across the entire page.

OffsetCol
Formats the text using an indented, one-column layout. The headings format across the page, with the offset text, or indented from the margin.

TwoCol
The text formats in two columns. Headings format across the page or with the two-column text.

You can vary the page layout at chapters and chapter-like elements. Just like on the IBMIDDoc tag, you use the Layout attribute on these elements: D, lers, preface, abstract, soa, legend, abbrev, bibilog, glossary, safety, IBMsafety, PNINDEX, and MasterIndex.

Setting the IBM copyright

The dates for the IBM copyright are taken from the IBMCopyr attribute. You typically enter either a single year or two years separated by a comma. Here's the setting for a new book, published for the first time in 1999:

```
<ibmiddoc ibmcopyr="1999">
```

Here's the setting for a book that was originally published in 1985, and was last published in 2000:

```
<ibmiddoc ibmcopyr="1985, 2000">
```

The date is printed in the edition notice; see "Notices and Edition notices" on page 98.

Setting the security classification

Sometimes your document needs to be confidential. To make it so, set the IBMIDDoc tag's IBMSec attribute to IC.

You can also enter the security and other information in the SEC attribute. For example, this shows the document as being confidential for company ABC. It also includes a date and time stamp; which is very useful when you are preparing multiple drafts for your reviewers.

```
<ibmiddoc sec="ABC Confidential - &date; &time;">
```

Setting page numbering to sequential or folio-by-chapter

Page numbering by chapter (known as folio-by-chapter) is a technique of numbering where the chapter number (or in the case of appendixes, the appendix letter) is prefixed to the page numbers, the figure numbers, and the table numbers. Thus, the fifth page in chapter 3 is numbered 3-5, the second figure in chapter 7 is numbered 7-2, and the fourth table in appendix E is numbered E-4. The page numbers, figure numbers, and table numbers are reset to 1 at the beginning of each chapter.

Adding the `PageNumber=FBC` attribute to the IBMIDDoc tag gives you folio-by-chapter page numbering.

If your book has more than one volume, see "Creating multiple volumes for a book".

Creating multiple volumes for a book

Sometimes you have a book that is just huge. To have the book printed, you need to divide it into volumes. To do this, you need to add the `MULTIVOL=Index-Folio` attribute to the IBMIDDoc tag. You can use either sequential or folio-by-chapter page numbering; see "Setting page numbering to sequential or folio-by-chapter" on page 85.

The Index-Folio setting adds `X-` as a prefix for the page numbers in the index and starts the page numbering from 1.

The remaining steps to create a set of multiple volume books are as follows:

1. You format your book as one large book, to get the table of contents, cross-reference, and index page numbers correct.
2. If your multi-volume books also have different cover pages and possibly order numbers, you will need to create mini-documents with that cover information.
3. You then use Adobe Acrobat to separate your large document into pieces; adding the separate pieces to the mini-documents.

A typical set of volumes would contain:

1. Volume 1
   a. cover
   b. edition notice
   c. table of contents, figure list, table list (for entire set)
   d. chapter 1 through 10, for example
   e. index (for entire set)
2. Volume 2
   a. cover
   b. edition notice
   c. table of contents, figure list, table list (for entire set)
   d. chapter 11 through 20, for example
   e. index (for entire set)

Controlling generated chapter, part, and appendix titles

Sometimes you may want to change the way "Chapter" or "Appendix" is added to your headings. The IBMIDDoc tag attributes PartPrefix ChapPrefix AppPrefix have attributes that allow you to control the form of this generated text. The have the basic form show below:

Text-Part, Text-Chap, or Text-App
This is the default. This outputs: "Part 1." for part headings, "Chapter 1." for chapter headings, or "Appendix A." for appendix headings.

Numonly-Part, Numonly-Chap, or Numonly-App
This is omits the word from the number. This outputs: "1." for part headings, "1." for chapter headings, or "A." for appendix headings.

None-Part, None-Chap, or None-App
This is omits the word and the number from the heading. Only the heading text itself appears.

Specifying the language of the document

The Language attribute specifies the language in which a document is written. It also specifies how the IBMIDDoc-generated text should be printed.

The valid values for the Language attribute on IBMIDDoc element are:

- BDUTCH or nl_BE
- BFRENCH or fr_BE
- BPORTUGUESE or pt_BR
- BULGARIAN or bg_BG
- CATALAN or ca_ES
- CENGLISH or en_CA
- CFRENCH or fr_CA
- CROATIAN or hr_HR
- CZECH or cs_CZ
- DANISH or da_DK
- DUTCH or nl_NL
- ENGLISH, en_US, or USENGLISH
- ESTONIAN or et_EE
- FINNISH or fi_FI
- FRENCH or fr_FR
- GERMAN or de_DE
- GREEK or el_GR
- HUNGARIAN or hu_HU
- ICELANDIC or is_IS
- ITALIAN or it_IT
- JAPANESE or ja_JP
- KOREAN or ko_KR
- LATVIAN or lv_LV
- LITHUANIAN or lt_LT
- MACEDONIAN or mk_MK
- NORWEGIAN or no_NO
- POLISH or pl_PL
- PORTUGUESE or pt_PT
- ROMANIAN or ro_RO
- RUSSIAN or ru_RU
- SCHINESE or zh_CN
- SERBIAN or sr_SP
- SFRENCH or fr_CH
- SGERMAN or de_CH
- SITALIAN or it_CH
- SLOVAK or sk_SK
- SLOVENIAN or sl_SI
- SPANISH or es_ES
- SWEDISH or sv_SE
- TCHINESE or zh_TW
- THAI or th_TH
- TURKISH or tr_TR
- UKENGLISH or en_GB

Bookmarks for PDF tables of contents

Adobe Acrobat PDF documents can have bookmarks generated that match the document's table of contents. This creates a very usable method of navigating the PDF file. The Xyvision PostScript formatter automatically adds these bookmarks.

Licensed and restricted materials

If you have a licensed or restricted document, use the CLASSIF attribute to indicate this. It is not within the scope of this book to explain what classification category is to be used for a particular document. The attribute looks like this:

CLASSIF= CONFRES | RES | LIC
: Identifies the classification of restricted materials.

CONFRES
: Confidential restricted material

RES
: Restricted material

LIC
: Licensed material

CLASSIF=LIC for the style TIV8x11, causes that Tivoli style to include the licensed statement on each page and on the cover.

Line justification for DBCS languages

This is used only for DBCS (double-byte character set) languages. While this is not supported at this time, you can have it in the source. This is used for left and right justification of text; the preferred format for DBCS languages. You specify the justification control this way:

```
<ibmiddoc style="xpp:(justify)">
```

The default is to not justify; the values `nojustify` and `ragged` indicate that.

About the prolog

The prolog is where we put information (marked up with special tags) about the whole document. For instance, the title, the author's name and address, and so forth.

Here is a simple prolog:

```
<prolog>
<ibmbibentry><doctitle><titleblk>
<title>My Cute, Little Document</title>
</titleblk></doctitle>
<ibmdocnum>SC99-1234-01</ibmdocnum>
<authors>
<author><person>
<name>Fred Mertz</name>
<address>East Overshoe, SD</address>
</person></author>
</authors>
</ibmbibentry>
</prolog>
```

Not all the tags you can use in the prolog are shown here. We'll discuss the ones shown first and then tell you about the others. The prolog itself does not cause anything to be printed; instead, it is a place to collect information that will be used in other places — for example, on the draft title page when the TIPAGE value is used.

Document title

The document's title is contained in the IBMBibEntry and Title elements. If the document title is short, you enter it with the Title element alone. If you want multiple lines in your title, you use the Title element and insert your lines as you want them formatted; putting a carriage return after each line to be split.

So a short title would be entered like this:

```
<title>Tom Sawyer</title>
```

whereas a long title would be entered like this:

```
<title>The Do's and Don'ts of
Caring for Your Fruit Bat</title>
```

If your title is very long, you may also want to use the STITLE element to get a short title (used in some document styles for the even-page running foot). For example:

```
<title>The Do's and Don'ts of
Caring for Your Fruit Bat</title>
<stitle>Fruit Bat User's Guide</stitle>
```

To have the title page appear in your document, see “Front matter (FrontM)” on page 98.

IBMBibEntry contains all of the bibliographic information for the document. It can contain the following:

- Authors
- DocTitle
- FileNum
- IBMDocNum
- IBMPartNum
- ISBN
- PublicID
- Publisher
- RetKey
- CoverDefs (“Adding to the front or back cover (CoverDef)” on page 91)

Document number

You enter the document number, if you have one, with the IBMDocNum element; after the document title. For example:

```
</doctitle>
<ibmdocnum>SC99-1234-01</ibmdocnum>
```

The last two digit-number is commonly called the “dash-level” and indicates how many times the book as been revised. This is always a two-digit number.

Author and Address

The Author and Address elements contain the author’s name and address information. The address is entered using as many lines as needed, surrounded by Address and its end tag. The text for each address line must be on a single line. These go within the Authors tag; just before the ending IBMBibEntry tag. For example:

```
<authors>
<author><person>
<name>Fred Mertz</name>
<address>
127 East Main Street,
East Overshoe, SD <postalcode>59134</postalcode>
</address>
</person></author>
</authors>
</ibmbibentry>
```

Date

The CritDates element contains a date for your document. For example, this specifies a date of September 9th, 2000:

```
</ibmbibentry>
<critdates>
<critdate>
<date>September 9th, 2000</date>
<desc>Date of publishing.</desc>
</critdate>
</critdates>
```

The current date prints on the draft title page. This critdate does not change the setting of the date symbol. The Critdate goes within Critdates; which comes after IBMBibEntry. Currently, it does not have much use.

Improving the searching of PDF books

Several prolog items will help in the focused searching of Adobe Acrobat PDF books. Ensure you have these items; they are passed through to the PDF document's "Information" dialog:

IBMIDDoc Element	**PDF Document Information field**
Library title (or stitle): document title (or stitle)	Title
Desc in IBMBibEntry	Subject
IBM, Tivoli, or blank (from company attribute on IBMIDDoc)	Author
Retkey in IBMBibEntry (after ending DocTitle)	Keywords
"XPP"	Creator
(blank -- automatic by Distiller)	Producer
(current date)	Created
(blank -- automatic by Distiller	Modified

Other prolog elements

The Prolog contains all tracking and control information for the document. Prolog can contain a number of elements, including:

- Approvers (similar to Authors)
- BibEntryDefs (Chapter 14, "Bibliographies and citations" on page 141)
- CopyrDefs ("Using CopyRDefs" on page 91)
- GLDefs ("Using GLDefs" on page 96)
- IBMProdInfo, IBM product information ("Using IBMProdInfo" on page 92)
- IDXDefs (Chapter 11, "Indexing" on page 115)
- LDescs (Chapter 12, "All about linking" on page 129)
- Maintainer ("Using reader's comment form (RCF)" on page 103)
- MasterIndexInfo ("Creating a master index" on page 125)

- ObjLib ("Reusing elements from an object library" on page 191)
- Owners (similar to Authors)
- ProdInfo
- PropDefs
- QualifDefs, qualification definitions ("Qualifying information" on page 49)
- RevDefs defines the revisions and marks that can be used in the document ("Defining Revisions in the RevDefs Element" on page 109).

Adding to the front or back cover (CoverDef)

You use the CoverDefs element to define cover artwork for your book's front and back covers. It is contained in the IBMBIBENTRY; the title for your book. For example:

```
<!entity front1 system "front1.eps" ndata graphics>
<!entity back1  system "back1.eps" ndata graphics>
 ...
<prolog><ibmbibentry><doctitle><titleblk>
<title>Sample Cover</title>
</titleblk></doctitle>
<coverdef>
<frontcover><mmobj><objref obj="front1">
<textalt>System/X cover artwork</textalt>
</mmobj></frontcover>
<backcover><mmobj><objref obj="back1">
<textalt>System/X back cover artwork</textalt>
</mmobj></backcover>
</coverdef>
</ibmbibentry></prolog>
```

You can also add text to the front cover. Inside the FrontCover tag, insert the PBLK tag and any content that you want to appear on the cover. For example:

```
<frontcover>
<pblk style="lblbox"><title>Notice</title>
<p>
The IBM License Agreement for Machine Code is included in this book.
Carefully read the agreement. By using this product you agree to
abide by the terms of this agreement and applicable copyright laws.
</p>
</pblk>
</frontcover>
```

Using CopyRDefs

If you have copyrighted material from some other company, you need to enter that company's information in the document prolog.

The CopyRDefs element contains the primary copyright information for the document. Use the CopyR element to specify this primary copyright information.

The primary CopyR element must have an ID attribute. This ID is referred to on the IBMIDDOC element using the Copyr attribute, as shown in the example that follows.

```
<IBMIDDOC COPYR="ibmprim">
 <PROLOG>

   ⋮
   <COPYRDEFS><COPYR ID="ibmprim">
    <P>IBM Corporation
       1994, 1995
       All Rights Reserved</P></COPYR>
   </COPYRDEFS>
   ⋮
 </PROLOG>
   ⋮
</IBMIDDOC>
```

Using IBMProdInfo

The IBMProdInfo element contains the IBM-specific product information about the product described in the document.

- ProdName, product name
- Version
- Release
- ModLvl, modification level
- IBMPgmNum, IBM program number
- IBMFeatNum, IBM feature number

For example:

```
<ibmprodinfo>
 <prodname>System/36</prodname>
 <version>2</version>
 <release>3</release>
 <modlvl>1</modlvl>
 <ibmprognum>223-3330</ibmprognum>
</ibmprodinfo>
```

Using Property Definitions (PropDefs)

PropDefs contains elements that define properties that can be used by other elements in your document. These properties apply to elements contained within the document or division with which the property definitions are associated.

All property definition elements can be used in PropDefs. These elements include:

- ClassDef ("Defining Element Classes" on page 202)
- PropDef ("Defining Element Properties" on page 201)
- LersDef (Chapter 17, "Defining Modular Information" on page 175)
- ModInfoDef (Chapter 17, "Defining Modular Information" on page 175)
- MsgItemDef ("Message and code lists" on page 38)
- PropGroup

PropDefs and Common Property Values

Use the PropDef element to define common property values. In the absence of ELETYPES or ID attributes, the property specified applies to all elements. These common properties are specified on common attributes. Examples of such common attributes are:

- Props
- Status
- Style

Limiting the Scope of PropDef Definitions

The global effect of PropDef definitions can be limited by specifying PROPNAME and ELETYPES. Scoping can also be limited by using a different set of PropDefs in each DProlog, instead of having only one set in the Prolog of the document.

For example, specifying ELETYPES='UL' on a PropDef element causes the other properties specified on the same PropDef element to apply to all UL elements.

The PROPNAME attribute with a value of `P001` provides a name to refer to when you want to apply the particular property definition to a specific element in your document using the PROPSRC attribute.

If both ELETYPES and PROPNAME are specified, the properties specified on the PropDef element apply to all of the specified element types, and may be referred to by the PROPNAME value.

If you wish that all figures be boxed figures, you can use the property definition described in the examples that follow.

```
   ⋮
<PROLOG>
   ⋮
<PROPDEFS>
 <PROPDEF PROPNAME="wider" ELETYPES="fig"
   STYLE="BKM:(width=page place=inline frame=rules)">
 </PROPDEF>
</PROPDEFS>
```

All Figure elements will now be framed.

In the next example, the PropDef has an ID of P001, and can be referenced by any element where such properties are valid.

```
   ⋮
<PROLOG>
   ⋮
<PROPDEFS>
 <PROPDEF PROPNAME="P001" ELETYPES="fig"
   STYLE="BKM:(width=page place=inline frame=rules)">
 </PROPDEF>
</PROPDEFS>
```

The document markup for a figure that includes PowerPC artwork would look like the example that follows.

```
<FIG ID="Unit" PROPSRC="P001">
 <FIGCAP>The IBM PowerPC CPU</FIGCAP>
 <MMOBJ>
  <OBJREF OBJ="ppcfig">

   ⋮
 </MMOBJ>
</FIG>
```

For hardcopy documents, you may have some column-wide figures. Instead of specifying the override on each column-wide figure, define your figure PROPDEF tags like this. The first PROPDEF (without the propname)sets the default for all figures; the second PROPDEF sets the column-wide override.

```
<propdef eletypes="fig" style="bkm:(place=inline)">
</propdef>
<propdef propname="colfig" eletypes="fig" style="bkm:(width=column place=inline)">
</propdef>
```

To get a column-wide figure, you specify ″COLFIG″ on the PROPSRC attribute of that figure. For example:

```
<fig propsrc="colfig">
```

By default, the BookMaster output process places these at the top of the next page. All other output processes place them inline. Use the INLINE override. For example:

```
style="bkm:(width=column place=inline)"
```

For wide figures in BookManager BOOKs, you may need to use the DWIDTH attribute. For example:

```
style="bkm:(width=column place=inline dwidth=100)"
```

For more information about using PropDefs, see Chapter 20, “Property and Class Definitions” on page 201.

Using PropDefs for Conditional Processing

Properties may be used to include or exclude information. This is called property-based retrieval.

For example, you can define a complex set of properties for doing conditional processing using PropDef, and then use those values for other elements by referring to the PropDef element, rather than having to explicitly specify those attributes on each element.

In the example that follows, a PropDef element is used to define the properties for including either RS6000 or PowerPC information.

```
<!DOCTYPE IBMIDDOC PUBLIC "+//ISBN 0-933186::IBM//DTD IBMIDDoc//EN"
 "ibmiddoc.dtd"
<IBMIDDOC COPYR="ibmprim"><PROLOG>
 </PROLOG>
  <PROPDEFS>
   <PROPDEF PROPNAME="ppc" PROPS="POWERPC #AND #NOT RS6000"
    <DESC>This PropDef will be referred to when PowerPC information is
     to be processed.</DESC>
   </PROPDEF>
   <PROPDEF PROPNAME="rs6" PROPS="RS6000 #AND #NOT POWERPC"
    <DESC>This PropDef will be referred to when RS6000 information is
     to be processed.</DESC>
  </PROPDEF>
 </PROPDEFS>
</PROLOG>
<BODY>
<D PROPSRC="PPC">
 <DPROLOG>
 <TITLEBLK>
  <TITLE>Installing a 128-Bit Video Card in the IBM PowerPC</TITLE>
 </TITLEBLK>
</DPROLOG>
 <DBODY><P>This procedure describes how to insert the 128-Bit
  video card in the IBM PowerPC.</P>
   <PROC>
     <TITLEBLK><TITLE>Installing a 128-Bit Video Card</TITLE></TITLEBLK>
      <PROCENTRY>

   ⋮
   </PROC>
  </DBODY>
 </D>

   ⋮
 <D PROPSRC="RS6">
  <DPROLOG>
   <TITLEBLK>
    <TITLE>Installing a 128-Bit Video Card in the RS6000</TITLE>
    </TITLEBLK>
   </DPROLOG>
   <DBODY><P>This procedure describes how to insert the 128-Bit
    video card in the RS6000.</P>
    <PROC>
     <TITLEBLK><TITLE>Installing a 128-Bit Video Card</TITLE></TITLEBLK>
      <PROCENTRY>

   ⋮
    </PROC>
  </DBODY>
 </D>

   ⋮
 </BODY>

   ⋮
</IBMIDDOC>
```

Using LDescs and Nameloc

LDescs contains elements to describe links and the locations used in these links.

Contained location elements can:

- provide an indirect reference to another SGML element. This protects the link(s) from changes made to the target element.
- associate a single ID with multiple elements
- support references to other documents, and to elements in other documents
- support references to non-SGML information.

For example, to link to multiple elements in the same document, you can use the Nameloc element to contain a list of IDs that are used to identify the information topic. In the example that follows, the LINKEND attribute refers to the Nameloc element with the `ID=DDInfo`. The elements in the document that have the `ID=aboutDan` and `ID=REHero` are linked to any link element which references the ID `DDInfo` using the LINKEND attribute.

```
<IBMIDDOC>
 <PROLOG>
  <IBMBIBENTRY>
   <DOCTITLE>
    <TITLEBLK><TITLE>Our Saturday Heroes</TITLE>
    </TITLEBLK>
   </DOCTITLE>
  </IBMIBENTRY>
 <LDESCS>
    <NAMELOC ID="DDInfo">
     <NMLIST>AboutDan REHero</NMLIST>
    </NAMELOC>
</PROLOG>
 <BODY>
  <D>
  <DPROLOG>
   <TITLEBLK><TITLE>Movie Serials</TITLE>
   </TITLEBLK>
   </DPROLOG>
  <DBODY>
   <P>The <L LINKEND="ddinfo">Dan Danger serial hero</L>
     was very popular in the 1940s.</P>
    <D>
    <DPROLOG>
     <TITLEBLK><TITLE>Male Heros</TITLE>
     </TITLEBLK>
    </DPROLOG>
   <DBODY>
    <P ID="aboutdan">Looking back, we now see Dan Danger as the
    quintessential Saturday morning serial hero.</P>
    <D>
     <DPROLOG ID="REHero">
      <TITLEBLK><TITLE>Heroes and Villains</TITLE>
      </TITLEBLK>
     </DPROLOG>
      <DBODY><P>More stuff about heroes....</P>
      </DBODY>
    </D>
   </DBODY>
  </D>
 </DBODY>
</D>
</BODY>
</IBMIDDOC>
```

Using GLDefs

Use GLDefs to contain GLEntry elements that can be used by reference from anywhere in your document. For more information about using GLEntry, see Chapter 13, "Glossaries" on page 137.

In the next example, terms are defined in GLDefs in the Prolog, and referred to by CONLOC reference within the document.

```
<PROLOG>
   ⋮
 <GLDEFS>
  <GLENTRY>
   <TERM ID="mainec">Maine Coon</TERM>
   <DEFN>A friendly and gentle breed of cat.</DEFN>
  </GLENTRY>
  <GLENTRY>
   <TERM ID="ragdoll">Rag Doll</TERM>
   <DEFN>A gentle breed of cat that may be even
         more docile than the Maine Coon.
   </DEFN>
  </GLENTRY>
 </GLDEFS>
</PROLOG>
<BODY>
 <D>
  <DPROLOG>
   <TITLEBLK>
    <TITLE>Movie Serials
    </TITLE>
   </TITLEBLK>
  </DPROLOG>
  <DBODY>
   <P>The <L LINKEND="ddinfo">Dan Danger serial hero</L>
      was very popular in the 1940s.  Dan's sidekick was a
     <TERM CONLOC="mainec"> named Elvis.</P>

   ⋮
</IBMIDDOC>
```

Using BibEntryDefs

BibEntryDefs contains BibEntry, LibEntry, IBMBibEntry, and IBMLibEntry elements.

An IBMIDDoc document requires a IBMBibEntry element, which contains the bibliographic information about the document.

BibEntryDefs is an optional Prolog element that can contain a variety of bibliographic entries that can be referred to throughout the document, or to build a bibliography by reference.

In the example that follows, BibEntryDefs contains several IBMBibEntry elements. The first IBMBibEntry is used to contain information about the containing document. The other IBMBibEntry elements contain entries for other referenced documents, and an IBMBibEntryDef that uses the CONLOC attribute to get its content from the containing document's IBMBibEntry. The example also includes an IBMLibEntry element.

```
<PROLOG>
<IBMBIBENTRY ID="BOOK0">
 <DOCTITLE>
  <TITLEBLK><TITLE>IBMIDDoc USER'S GUIDE</TITLE></TITLEBLK>
 </DOCTITLE>
 <AUTHORS>
  <AUTHOR>
   <PERSON>
    <NAME>FRED MERTZ</NAME>
    <ADDRESS>
     <INTERNET>fredm@usa.ibm.com</INTERNET>
    </ADDRESS>
   </PERSON>
  </AUTHOR>
 </AUTHORS>
 <IBMDOCNUM>SH21-0783-02</IBMDOCNUM>
</IBMBIBENTRY>
   ⋮
  <BIBENTRYDEFS>
   <IBMBIBENTRY ID="BOOK1">
    <DOCTITLE>
     <TITLEBLK><TITLE>IBMIDDoc MIGRATION GUIDE</TITLE>
     </TITLEBLK>
    </DOCTITLE>
   </IBMBIBENTRY>
   <IBMBIBENTRY ID="BOOK2">
    <DOCTITLE>
     <TITLEBLK><TITLE>IBMIDDoc REFERENCE GUIDE</TITLE></TITLEBLK>
    </DOCTITLE>
   </IBMBIBENTRY>
   <IBMBIBENTRY ID="BOOK3">
    <DOCTITLE>
     <TITLEBLK><TITLE>IBMIDDoc TUTORIAL </TITLE></TITLEBLK>
    </DOCTITLE>
   </IBMBIBENTRY>
   <IBMLIBENTRY ID="IDDOCLIB">
    <LIBRARY>
     <TITLEBLK><TITLE>IBMIDDOC</TITLE></TITLEBLK>
    </LIBRARY>
    <PUBLISHER>
     <CORPNAME>IBM</CORPNAME>
    </PUBLISHER>
    <CONTAINEDDOCS BIBIDS="BOOK1 BOOK2 BOOK3">
   </IBMLIBENTRY>
  </BIBENTRYDEFS>
</PROLOG>
```

For more information about IBMIDDoc bibliographic elements, see Chapter 14, "Bibliographies and citations" on page 141.

Front matter (FrontM)

The front matter contains the title page, notices (such as the edition notice), the preface, the summary of changes, the table of contents, the table list, and the figure list. The elements that can be used in FrontM include:

- EdNotices (see "Notices and Edition notices")
- TOC (see "Table of contents" on page 99)
- FigList (see "List of figures" on page 100)
- TList (see "List of tables" on page 100)
- Preface (see "The preface" on page 100)
- SOA (see "Summary of changes" on page 100)
- Abbrev (abbreviations), Abstract, Legend, and others (see)
- D, divisions
- IBMSafety (see "IBM Safety text" on page 101)
- Safety (see "IBM Safety text" on page 101)

The FrontM Style attribute can specify the following values using the Display keyword. You specify these in any combination.

TIPAGE
Causes a draft title page to appear (this should not be used for final camera-ready output).

COVER
Causes a cover, inside title page, and back cover page to appear.

SPINE
Causes the spine to appear after the back cover. The spine contains the IBM Logo, Library, Title, and Version number.

OLDSPINE
Causes the spine to appear after the back cover. The spine Includes the IBM Logo, Library, Title, Version number, and document number.

NORECYCLE
Prevents the recycle logo form appearing on the back cover. Removes the recycled paper logo and text from the back cover of a US English document

REGLOGO
This uses a registered logo on the back cover even when the PrtLoc element is used in the document.

For example, this causes a draft title page, the cover, inside cover, back cover, and spine to be output as part of your document:

```
<FRONTM STYLE="display='TIPAGE COVER SPINE'">
```

Notices and Edition notices

Anything you want to put on the back of the title page are known collectively as "notices". Some documents have only an edition notice, which goes at the bottom of the back of the title page; others have notices in addition to the edition notice.

The edition notice involves the EDNotice tag and the Title tag. The Title tag is used immediately following the EDNotice tag, and specifies the text of the heading for the edition notice. A sample edition notice might look like this:

```
<ibmiddoc ibmcopyr="1996, 1999">
 ...
<ednotices><title>First Edition (June 1997)</title>
<p>This edition applies to the IBMIDDoc language,
Version 4.2, and to all subsequent releases
and modifications until otherwise indicated in new
editions.</p>
</ednotices>
```

Look at the page following this book’s title page to see what IBMIDDoc does with the edition notice. The EDNotice end tag brings in the copyright line, if the IBMCopyr attribute was specified on your IBMIDDoc tag (or if the COPRNOTE tag is used).

Other notices

If you have other things to put on the back of the title page besides the edition notice, put them all within a NOTICES tag and its matching end tag before the EDNotice tag (if there is one). It might look like this:

```
<notices><pblk style="lblbox"><title>Note</title>
<p>Before using this information, be sure to read
the general information under <xref refid="notices">.
</p>
<p>This manual was produced using IBMIDDoc SGML, the
Epic editor, and processed for print and online using
the ID Workbench.</p>
</pblk></notices>
<ednotices>
```

The NOTICES tag (with its end tag) is actually allowed anywhere in your document. If you put it before the EDNotice tag, the notice associated with it appears on the back of the title page.

Table of contents

The TOC contains the table of contents the document. You can choose to use the GendTitle, for which the title text is generated automatically, or you can enter your own title for this special division by using the Title element.

Typical markup for a TOC:

```
<toc><gendtitle></toc>
```

A TOC with a heading you’ve specified:

```
<toc><titleblk><title>
Here's what's in my cool, little booklet
</title></titleblk></toc>
```

You can control which heading levels appear in the table of contents by using the MAXTOC attribute on the IBMIDDoc tag. For example, the default for the style IBM8X11 is to show headings in the table of contents to heading level 3. Specifying the following will include divisions to heading level 4 to appear:

```
<ibmiddoc maxtoc="4" docstyle="ibm8x11">
```

You can also control which headings appear in a table of contents by using the TOC attribute on the Division or other heading tag.

| To create a partial table of contents for a part or chapter; see “Partial table of
| contents” on page 24.

List of figures

Use the FigList element to contain a list of figures that appear in the document. You can choose to use the GendTitle, for which the title text is generated automatically, or you can enter your own title for this special division by using the Title element.

Typical markup for a FigList:

```
<figlist><gendtitle></figlist>
```

List of tables

Use the TList element to contain a list of tables that appear in the document. You can choose to use the GendTitle, for which the title text is generated automatically, or you can enter your own title for this special division by using the Title element.

Typical markup for a TList:

```
<tlist><gendtitle></tlist>
```

The preface

Use the Preface element to contain explanatory or preparatory information about the document. As with other FrontM elements, you may use the GendTitle element or provide a unique title by using the TitleBlk element. Enter the preface text using the same rules you follow when creating any other division.

```
<preface>
 <specdprolog><gendtitle></specdprolog>
 <dbody>
  <p>This manual...</p>
 </dbody>
</preface>
```

If you don't want to use the generated title, you can enter another title within a TitleBlk element.

```
<preface>
 <specdprolog><titleblk>
  <title>About this book</title>
  </titleblk></specdprolog>
 <dbody>
  <p>This manual...</p>
 </dbody>
</preface>
```

Summary of changes

Use the SOA element to contain a list or description of the important information that has been changed or added since the last revision of the document. SOA is valid in FrontM (recommmended) and BackM. For example:

```
<soa>
 <specdprolog><titleblk><title>What's new and different
  </title></titleblk></specdprolog>
 <dbody>
  <p>Changes since the last edition include...</p>
 </dbody>
</soa>
```

Special sections

There are a number of special sections that often occur in publications, typically in either the front matter or the back matter. IBMIDDoc recognizes these special sections (as well as the preface, which we've already covered):

Special Section	**Tag**
List of Abbreviations	ABBREV
Abstract	ABSTRACT
Bibliography	BIBLIOG
Legend	LEGEND

IBM Safety text

Use IBMSafety to contain any IBM-specific safety concerns or issues that are addressed in your information.

```
<FRONTM>
 <IBMSAFETY SPEC="AUTO">
  <GENDTITLE>
 </IBMSAFETY>
</FRONTM>
```

Note: Not supported by Xyvision.

About back matter (BackM)

The BackM element can contain Appendicies, the bibliography, glossary, index, part number index, and Divisions.

The Xyvision and BookMaster transforms provide a part separator for the back matter when the body of the document contained a PART tag. If you want to suppress the automatically-generated part separator, use the following coding on the BACKM tag:

```
<backm style="xpp:(nopart)">
```

Using appendix

The Appendix element contains divisions that contain appendix information. Appendix is valid in BackM.

You must enter titles for the appendixes. For example:

```
<BACKM>
 <APPENDIX>
  <D>
   <DPROLOG>
    <TITLEBLK>
     <TITLE>Whantoozler Tuning Parameters</TITLE>
    </TITLEBLK>
   </DPROLOG>
   <DBODY>
    <P>There are many settings that you can adjust to
       improve Whantoozler performance.
    </P>
   </DBODY>
  </D>
 </APPENDIX>
</BACKM>
```

Using glossary

Use the Glossary element to contain a list of the glossary terms for the document. You can choose to use the GendTitle, for which the title text is generated automatically, or you can enter your own title for this special division by using the Title element. Glossary should contain a GL element, which can contain an explicit list of GLEntry elements, or can use the AUTO value on the SPEC attribute. The AUTO value on the SPEC attribute causes the GL to contain a list of all the GLEntry elements in the document. AUTO is the default value for the SPEC attribute, and is the usual way to create a list of GLEntry elements in a GL.

```
<BACKM>
 <GLOSSARY>
  <SPECDPROLOG>
   <GENDTITLE>
  </SPECDPROLOG>
  <DBODY>
   <GL> ... </GL>
  </DBODY>
 </GLOSSARY>
</BACKM>
```

See Chapter 13, “Glossaries” on page 137 information about glossary.

Using bibiography (Bibliog)

Use the Bibliog element to contain a list of documents related to the document. Bibliog is valid in both FrontM and BackM.

Bibliog should contain the BibList element, which can contain an explicit list of BibEntry elements.

You can choose to use the GendTitle, for which the title text is generated automatically, or you can specify the title of the Bibliog (″Related Publications″) using the TitleBlk elements. For example:

```
<bibliog>
<specdprolog><gendtitle></specdprolog>
<dbody>
<biblist><bibentry><doctitle><titleblk><title>My Nice
Book</title></titleblk></doctitle></bibentry>
<bibentry><doctitle><titleblk><title>Your Nice Book
</title></titleblk></doctitle></bibentry>
</biblist>
</dbody></bibliog>
```

See Chapter 14, “Bibliographies and citations” on page 141 for more information.

Using part number index (PNIndex)

A Part Number Index can be automatically generated by including the PNIndex element in the BackM element. In most cases, you will use the GendTitle element to include the system-defined title for the PNIndex.

```
<PNINDEX><GENDTITLE></PNINDEX>
```

For more information on these, see Chapter 23, “Creating parts catalog lists” on page 215.

Using Index

The Index element content is normally generated automatically at processing time from the index tags you’ve sprinkled throughout the source.

```
<INDEX><GENDTITLE></INDEX>
```

For more information about creating indexes, see Chapter 11, “Indexing” on page 115.

The Index should be the last item that has content in a document. Only the reader's comment form should follow the index.

Using reader's comment form (RCF)

A Reader Comment Form can be automatically generated by including the RCF element in the BackM element. In most cases, you will use the GendTitle element to include the system-defined title for the RCF. There should no longer be a "Contacting IBM" section before the RCF; that now belongs in the preface.

For the RCF to be generated, you need to specify the MAINTAINTER element information in the prolog of your document:

```
<maintainer>
<corp>
<corpname>IBM Corporation</corpname>
<address>ATTN: Dept 542
3605 HWY 52 N
Rochester, MN
<postalcode>55901-9986</postalcode>
<phone equip="fax">1-800-555-1212</phone></address>
</corp>
</maintainer>
```

In the back matter, you need to include the RCF element; for example:

```
<backm>
<rcf><gendtitle></rcf>
</backm>
```

Chapter 9. Using definition tags

There are a number of elements that occur in IBMIDDoc documents that have these properties:

- There are many things to say when describing the elements (that is, they have many attributes).
- The element is likely to occur many times in the same document, using the same (or nearly the same) set of attributes.

We would all like to save time in marking up these documents. Particularly, we'd like to save time when we change our minds about the values of the attributes we want to use. Wouldn't it be nice to be able to do this in one place rather than search through the document and change it in all places that it occurs?

To allow you to describe things once and then use the descriptions repeatedly, IBMIDDoc provides a number of "definition" tags. They are all recognizable because their names end with the letters "DEF." The tags and their corresponding definition tags are shown in Table 15.

Table 15. Tags and corresponding xxDEF tags

Element	**Corresponding DEF Element**	**xxDEF Tag Attributes**
DL	DLDEF	DefName, Props, LineSpace, TermWidth, TermStyle, HeadStyle
FIG	FIGDEF	DefName, Props, Frame, ScalePct, PgWide
GL	GLDEF	DefName, Props, LineSpace, RetKey
LERS	LERSDEF	DefName, Props, Sep, Retkey, Auth, Comments, Context, Defaults, ErrCond, Examples, Flags, Format, Intrep, Messages, Other, Parms, Process, Purpose, Restrict, Results, Retcodes, SysEnv, Usage, Version, ClassName, Conloc, ID, Rev, Status, Style, HyTime, RefType, InfoMast
MSGLIST	MSGLDEF	DefName, Props, Layout, RetKey
NOTELIST	OLDEF	DefName, Props, LineSpace. An OLType attribute from any OLDEF tag is ignored because NoteLists have no OLType attribute..
OL	OLDEF	DefName, Props, LineSpace, OLType
PARML	DLDEF	DefName, Props, LineSpace, TermWidth, TermStyle, HeadStyle
SCREEN	SCREENDEF	DefName, Props, PgWide LineLength
SYNTAX	SYNTAXDEF	DefName, Props, SynStyle, ScalePct, PgWide
UL	ULDEF	DefName, Props, LineSpace, ULType
XMP	XMPDEF	DefName, Props, PgWide

We're not going to describe all of these tags and their attributes here; we just want you to understand how definitions work so that we don't have to explain it each time we describe one of the tags above. The definition tags set the common attributes of the using tags. Once your specify the attribute on the definition tag, it is not necessary to specify it again on the using tag. Definition tags have a DefName attribute, which is the means by which a using tag asks for a particular definition with a matching DEF attribute. Definition tags can be specified without a DefName attribute, in which case they essentially replace the initial settings of attributes for the entire document.

In all IBMIDDoc definitions, the using tag (that is, the tag with the DEF on it that uses the definition) can override individual attributes from the definition by specifying them again. You are not required to use a definition tag; but they sure make life easier.

The DEF tags go in your document's prolog, inside a PropDefs tag. That way they apply to your entier document. You can also put DEF tags in a division's DProlog tag (in a PropDefs tag); these only apply to the division and it's children.

Let's consider an example. The FIG tag, as initially shipped with IBMIDDoc, has the attribute initial settings shown here:

```
FRAME=none
PGWIDE=0
```

Let's say that for a document we need to work on, that most of the figures will have ruled frames and be page-wide. We would specify a FIGDEF tag in the Prolog with the following attributes. Note that there is no DefName attribute; this overrides the document's defaults.

```
<prolog>
...
<propdefs>
...
<figdef frame=rules pgwide=1>
```

However, a handful of our figures, will be column-wide, boxed, and have a slightly larger type size. This shows an additional FIGDEF to handle these figures; note that is has a DefName attribute::

```
<prolog>
...
<propdefs>
...
<figdef frame=rules pgwide=1>
<figdef defname=colfigs pgwide=0 frame=box scalepct=150>
```

Now most of the figures in our book can be described simply by using the FIG tag with no attributes (except the FIG tag's own ID for cross references), but the boxed figures will have to use a DEF=COLFIGS attribute as well. Then there is always one maverick that insists on being different from everyone else; this one suffers from having too much content and must be scaled down:

```
<fig id=abc>
...
<fig id=jkl def=colfigs>
...
<fig id=xyz def=colfigs scalepct=50>
```

Let's summarize what we did here: first, we redefined the FIG attribute initial settings for our use in this document (the FIGDEF tag with no DefName). Then, we set up the FIG attributes for some of our figures (the FIGDEF tag with a DEFNAME). Finally, we show our figure markup (1) using the new initial settings, (2) using the “colfigs” values, and (3) using the “colfigs” attribute values plus an override for the SCALEPCT attribute.

Here's how the figures appear:

Figure has a ruled frame and is page-wide.

Figure 8. "default"

Figure has a boxed frame, is column-wide, with larger type.

Figure 9. "colfigs"

Figure has a boxed frame, is column-wide, with smaller type.

Figure 10. "colfigs" with an override

Summarizing the initial setting override hierarchy

"Initial setting override hierarchy" is a fancy way of describing a simple process. IBMIDDoc has a set of initial values for attributes which can be overridden in your document by:

1. Definition tags without DefName attributes, which can be overridden by:
2. Definition tags with DefName attributes and using tags with matching DEF attributes, which can be overridden by:
3. The using tag itself.

Here's a complete description of how ID Workbench processing checks each attribute and determines its value:

1. An attribute placed on an element directly
2. An attribute placed on an element definition with defname='mydef', when the current element has the attribute def='mydef'
3. An attribute placed on a generic element definition for this element (for example, a FIGDEF or OLDEF with no defname).
4. An attribute placed on a classdef with classname='myclass', when the current element has the attribute class='myclass'
5. An attribute placed on a propdef with propname='myprop', when the current element has the attribute propsrc='myprop'
6. An attribute placed on a propdef with no propname.

Each of these describes the behavior for a single attribute. Every attribute is checked individually, in the order described above. This means, if a figure tag is defined as follows:

```
<fig def='mydef' pgwide=0>
```

The pgwide will override any other definitions, but all other attributes (such as frame, scalepct, or style) will follow the normal course of inheritance. This is the case for every attribute on every tag covered by the new DEF elements. For any other tags, the same order of precedence applies, except that numbers 2 and 3 are skipped.

On the LERSDEF tag, there are both DefName and CLASSNAME attributes. If both are specified, you get a warning in the transformation, and only the DefName attribute is used. Otherwise, the CLASSNAME and DefName attributes are considered identical, as are the DEF and CLASS attributes on the LERS tag. In step 2 of the attribute hierarchy above, any combination of classname/defname with the class/def attributes is valid. For example:

```
<lersdef defname='mydef'>
```

can be referenced either by

```
<lers def='mydef'> or <lers class='mydef'>
```

Similarly:

```
<lersdef classname='myclass'>
```

can be referenced either by

```
<lers def='myclass'> or <lers class='myclass'>
```

Chapter 10. Revision Elements and Marked Notes

IBMIDDoc has a slick way of indicating changes in your document. Use RevDefs, Rev, and Mark elements to track revisions in your documentation. These elements are valid in the Prolog or DProlog elements of your document. You can also use a marked deletion (MD) element to indicate some text you are going to remove.

Revision and Marked Notes elements include:

- Rev, define a revision level
- RevDefs, define several revisions
- REV attribute, assign a revision level to an element
- MD, indicate words or phrases the be removed; the text is formatted with a strike-through font

The following Mark elements are not yet fully supported; they only work for BookMaster output processing.

- Mark
- MkAction
- MKClass
- MKDesc
- MKNote
- MarkList

Using Revisions

In order to define revisions, you must place a Rev element in a RevDefs element. The RevDefs can be in your document prolog to affect the whole document; or in a division DProlog to affect only that division. The Rev element defines the specifics for a single revision that can be used throughout the document or division. You can have more than one revision in a document. This is useful for multiple drafts. Normally, in hardcopy, a revised text is indicated by a vertical bar to the left of the text. You can set different characters for your different revisions. Any element that contains new, changed, or deleted information can refer to the ID attribute value on the Rev element to denote why and how the information has changed.

Defining Revisions in the RevDefs Element

The RevDefs element contains several Rev elements. Each Rev defines a revision, describes the reason for the revision, and states if the revision should be used or ignored during document processing. If the Rev is to be used, the character specified on the char attribute will be printed in the margin to the left of the changed information. If the revision is ignored, no special characters will appear beside the information.

To begin, get to the prolog of your document and insert a RevDefs element; then insert a Rev element. Pick an identifier for your REV tag that makes sense. Suggestions include:

- v4r5 — this indicates version 4, release 5
- r2m1 — this indicates release 2 modification 1; or 2.1

Then, set the IDENT attribute to USE; this enables the revision. If you want to turn off the revisions, set IDENT to IGNORE.

If you want a special character to indicate the revision, type that one character into the CHAR attribute. Normally, a vertical bar is used.

Here is a simple revision definition; only one revision is defined in the document's prolog:

```
<prolog>
 ...
<revdefs>
<rev id="v4r5" ident="use">
<date>9/9/99</date>
<desc>First draft for v4r5</desc>
</rev>
</revdefs>
 ...
</prolog>
```

Here is a more complicated set of revision definitions. ″v4r5″ and ″v4r5d2″ are enabled, with characters ″|″ and ″+″ being used respectively. The other revision for ″v4r4″ is defined, but set to ignore.

```
<revdefs>
<rev id="v4r5" ident="use">
<date>9/9/99</date>
<desc>First draft for v4r5</desc>
</rev>
<rev id="v4r5d2" code="+" ident="use">
<date>9/10/99</date>
<desc>Second draft for v4r5</desc>
</rev>
<rev id="v4r4" ident="ignore">
<date>9/9/98</date>
<desc>V4R4 changes</desc>
</rev>
</revdefs>
```

| Currently, you can have up to 20 different revision levels active in a document
| being formatted for Xyvision PostScript or PDF.

Indicating Revisions in the Document Markup

To indicate that text is new or changed; you use the REV attribute on the element that contains the text. The REV attribute refers to the REV elements defined in the RevDefs. The revision characters start with the beginning tag, and continue through to the ending tag.

Using the ″v4r5″ revision definition from "Defining Revisions in the RevDefs Element" on page 109, the middle list item is changed:

- something old
- | something new (or changed)
- something borrowed

Here's its markup:

```
<ul>
<li>something old</li>
<li rev="v4r5">something new (or changed)</li>
<li>something borrowed</li>
</ul>
```

If you need to revise a section that is part of another revision, IBMIDDoc allows you to nest the revisions; that is, to place revisions inside other revisions. When the formatter encounters the REV attribute for your new revision, it stops printing the character associated with the old revision and starts printing the character you assigned to the new revision. Then, when the formatter encounters the end tag for your new revision, it resumes printing the character associated with the old revision.

For example, if the whole list was changed for ″v4r5″, then the middle item was added for the second draft; you would want something like this:

| • something old
+ • something new (or changed)
| • something borrowed
| • something blue

Here’s its coding:

```
<ul rev="v4r5">
<li>something old</li>
<li rev="v4r5d2">something new (or changed)</li>
<li>something borrowed</li>
<li>something blue</li>
</ul>
```

The revision markup doesn’t evaluate the dates of the revision definitions, it works on nested position. It’s totally possible to make a small change first and then make a larger change that encompasses the first change. The inner revision will still show its change bar. Watch out for these; you might need to delete the nested Rev attributes if they should really be part of outer revision.

Marking text for deletion

Sometimes, as part of a revision, you may want to indicate that some text has been deleted, but still leave that text in the document for your reader’s convenience. If you precede the text to be deleted with the MD (marked deletion) tag and follow it with the matching end tag, the formatter overstrikes the text with a horizontal line. The MD element is like a phrase. For example:

| You may want to eliminate ~~repetitious~~ redundancies.

Here’s its markup:

```
You may want to eliminate <md rev="v4r5">repetitious</md>
redundancies.
```

Creating Collections of Marked Notes

To use marked notes in IBMIDDoc, you enter mark elements that define the marked note collection. These are contained in the RevDefs element, which is contained in the Prolog or DProlog elements. You must define at least one collection in order to use marked notes in your document. These collections of notes of can be put in a number of forms, including a table.

Processing Note

Marked Notes currently only work in BookMaster output processing.

These are the elements you'll need to use to create a marked collection of notes:
- Mark
- MKAction
- MKClass
- MKDesc
- MkNote
- MarkList

Using the Mark Element

The Mark element names a marked collection of changes and specifies whether or not the other elements associated with this marked collection are processed when your document is formatted. Mark elements are contained in the RevDefs element, along with Rev elements.

A typical Mark element looks like the one in this example:

```
<mark id="mkv4r5" ident="use">
<desc>v4r5 marked message changes</desc>
</mark>
```

Defining Marked Actions and Classes

MkAction is used to define one or more actions that can be associated with marked notes. These actions can be used with any marked class. MkClass defines a marked note class within MkDesc. You can define as many classes as you need. These class codes are also used on the MarkList element to tell IBMIDDoc which class codes to make part of the marked notes table.

```
<propdefs>
<mkdesc>
<!--Define two classes for marked lists - notes and abends-->
<mkclass name="msg">Msg</mkclass>
<mkclass name="abend">Abend</mkclass>
<!--Define the actions for the changed info-->
<mkaction name="new">New</mkaction>
<mkaction name="change">Changed</mkaction>
<mkaction name="del">Deleted</mkaction>
<mkaction name="rep">Replaced</mkaction>
</mkdesc>
</propdefs>
```

Using the MkNote Element

The MkNote element identifies the actual text of your marked note. Several attributes are used on the MKNote element. These include:

CLASS
defines one or more mark classes to which the marked note belongs.

ACTION
defines one or more actions associated with the marked note.

MKIDS
contains the ID of one or more Mark elements.

ITEM
defines an identifying label for the note, such as a message number or error report.

The content of the tag displays in the description column of the marklist table.

```
<mknote class="msg" action="change" mkids="mkv4r5" item="IDW0012">Hi there!
</mknote>
```

Generating a Collection with MarkList Element

The MarkList element causes a table of marked collection notes to be generated. You can include any notes that you mark in the marked note list, and you can headings for the table. For example, the marked note list can be used to generate a definitive summary of changes. In addition, you can use marked notes to collect information about document content, notes to yourself or others, or references to certain locations in the document that you think will be very important to the reader.

The MarkList element generates a list of marked notes at the place in the document where the MarkList element is specified. Only notes of the specified classes, collections, and actions will be included in the generated list.

```
<marklist mkids="mkv4r5" classes="msg abend" actions="change del rep"
display="item action page desc" classhd="Msg" actionhd="Reason"
itemhd="Msg" lochd="Page" deschd="Message text">
```

A Marked Notes Markup Example

The example that follows illustrates how to use marked notes in IBMIDDoc.

```
<ibmiddoc docstyle="ibmxagd">
<prolog><ibmbibentry><doctitle><titleblk>
<title>My Marked Changes Document for Messages</title>
</titleblk></doctitle></ibmbibentry>
<propdefs>
<mkdesc>
<!--Define two classes for marked lists - notes and abends-->
<mkclass name="msg">Msg</mkclass>
<mkclass name="abend">Abend</mkclass>
<!--Define the actions for the changed info-->
<mkaction name="new">New</mkaction>
<mkaction name="change">Changed</mkaction>
<mkaction name="del">Deleted</mkaction>
<mkaction name="rep">Replaced</mkaction>
</mkdesc>
</propdefs>
<revdefs>
<rev id="revv4r5" ident="use">
<date></date>
<desc></desc>
</rev>
<mark id="mkv4r5" ident="use">
<desc>v4r5 marked message changes</desc>
</mark>
</revdefs>
</prolog>
<body>
<d>
<dprolog><titleblk>
<title>List of changed items</title>
</titleblk></dprolog>
<dbody>
<marklist mkids="mkv4r5" classes="msg abend" actions="change del rep"
display="item action page desc" classhd="Msg" actionhd="Reason"
itemhd="Msg" lochd="Page" deschd="Message text"></dbody>
</d>
<msglist>
<msg rev="revv4r5">
<msgnum>IDW0012</msgnum>
<msgtext>Hi there!</msgtext>
<msgitem class="xpl">
```

Revisions

```
<p>This is a friendly message.
<mknote class="msg" action="change" mkids="mkv4r5" item="IDW0012">Hi there!
</mknote></p>
</msgitem>
</msg>
<msg rev="revv4r5">
<msgnum>IDW0013</msgnum>
<msgtext>Farewell!</msgtext>
<msgitem class="xpl">
<p>This unlucky message was removed.
<mknote class="msg"
action="del" mkids="mkv4r5" item="IDW0013">Farewell!
</mknote></p>
</msgitem>
</msg>
</msglist></body>
</ibmiddoc>
```

The resulting marklist table will look like the example that follows.

Msg	**Reason**	**Page**	**Message text**
IDW0012	Changed	1	Hi there!
IDW0013	Deleted	2	Farewell!

Chapter 11. Indexing

Creating an index using IBMIDDoc is somewhat like creating a table of contents (TOC). The index is built for you from items in the text you have tagged as index entries. Just as you use the TOC tag to indicate you want a table of contents, you use the INDEX tag in the back matter of your document to show that you want the index included.

Indexes are similar to a TOC in that a subject and page number are listed. However, the index can provide much more detail than a TOC. It provides a term, subterms, and sometimes synonyms (in the form of “see and see also” references) with page numbers indicating where detailed information can be found on the subject. Indexes are also sorted alphabetically for easy subject retrieval.

You can build an index by tagging the terms you think will be useful for the reader. Place the index tags at the point that the topic occurs to ensure the page references in the index will be correct whenever you format the document. The formatter will automatically create an index in alphabetical order and place the correct page number next to each entry in the index.

Table 16 illustrates the terminology we use in this chapter to describe the elements of an index.

Table 16. Terminology used in discussion of indexing

entry:	appetizers 102
	⋮
subject:	bechamel sauce 13
page references:	cabbage 58, 115
	⋮
primary entry:	eggs 106
secondary entry:	souffles 108
tertiary entry:	chocolate 112
	⋮
entry heading:	meats
	beef 60
	⋮
	poultry 75
“see also” reference:	*See also* chicken, turkey
	⋮
	white sauce
“see” reference:	*See* bechamel sauce

In this chapter we will discuss the levels of indexes, where to place index entries, how to define index entries, refer to index entries, the use of see and see also references, and how to control and generate an index. But first we will discuss the basic index structure.

Structuring a basic index

A good index in an indispensable part of any document. This is especially true for reference documents. Because you don't usually read reference information from cover to cover, you need a way to be able to find specific bits of information you need.

To be a good, complete retrieval device, an index must do the following:

- Help readers find information within the document.
- Anticipate how readers will search for information.
- Serve the novice and the expert.
- Show how topics interrelate.
- Tell what the book contains.
- Cross-reference similar terms or concepts.

Before we talk about the actual tags, here are some good tips as you develop an index:

- Familiarize yourself with the content, organization, and objectives of the document before you start the indexing process.
- Analyze your audience. Who will be using the book? Are readers likely to be familiar with the book and the product? What will the reader already know?
- Ask yourself, “Does this topic contain information the reader will want to find?” If so, create at least one index entry for that topic.
- Develop an indexing worksheet for each section of your document. On it list major concepts or ideas, major terms defined, acronyms and abbreviations, restrictions and warnings, and cross-references to other information products. Use the worksheet to determine which topics should be main entries and which should be subentries. The worksheet also ensures that important information is not left out of the index.
- Be sure to use both the acronym or abbreviation and its “spelled-out version” as index entries if your document uses them.
- Ask yourself when looking at an index entry, “Are there any commonly used synonyms for this word?” If so, include them in your index as well.
- Make sure that each index entry has no more than two or three references. Use specific subentries to reduce the number of page references and give your reader a more precise pointer to the topic.

The IBMIDDoc indexing elements include:

- I1, primary
- I2, secondary
- I3, tertiary
- IdxTerm, index term text
- IRef, index reference
- IdxDefs, index definitions
- Index, index placement

Basic index tagging

There are 3 levels of index entries: Primary (i1), Secondary (i2), and Tertiary (i3). The simplest kind of index entry is a primary entry. A primary entry is the major subject and should be a noun or noun phrase. A primary entry with a page number, is entered with the I1 tag, which says, “This is an index subject at the first level”. It would look like this:

```
<i1><idxterm>dessert sauces</idxterm></i1>
```

A primary index entry may or may not have page number listed. However, if the primary index entry does not have a secondary entry associated with it, the primary tag will automatically have a page number entered. There will be more on index entries and page numbers later in this chapter.

Most indexes run to more elegant structures with primary, secondary, and sometimes tertiary entries. The secondary entry narrows the primary entry into a more specific subject. It may or may not have a page reference. Secondary entries are arranged alphabetically in the index following the primary entry to which they apply. A secondary entry with a page number is entered with the I2 tag.

When you have a large number of subtopics under your primary entry, a secondary or tertiary tag improves index readability. If you’ve ever seen an index where most of the entries are primary and have page numbers, you know how difficult it can be to find the information you need. Using the I2 tag makes an entry stand out and directs the reader’s attention to a topic instead of a mass of numbers.

Tertiary entries are the third, even more specific level for the major topic. A tertiary entry always has a page reference. Tertiary entries are arranged alphabetically following the secondary entry to which they apply.

Placement of index tags

Indexing isn’t as easy as tossing an i1 tag here and an i2 tag there. Believe it or not, there are “rules” for index tagging unless you want an index with only primary index entries. As we discussed before, secondary and tertiary entries make an index more readable. So, unless you have a small index, you’ll want to add a few index levels.

There are two ways to associate secondaries with their primaries and tertiaries with their secondaries. One way is by their position in the source file. The other way is by creating cross references. We’ll tell you all about the position way first.

Position method

The position method has the secondary entries within the tags of the primary entry. Likewise, the tertiary entries are embedded in the secondary entries. The rule when using the position method is you cannot have the secondary entries listed outside the primary entry and the tertiaries cannot be outside the secondary entry. Here’s an example of the position method:

```
<i1><idxterm>dessert sauces</idxterm>
<i2><idxterm>butterscotch</idxterm></i2>
<i2><idxterm>hot fudge</idxterm>
<i3><idxterm>microwave method</idxterm></i3>
```

```
<i3><idxterm>stovetop method</idxterm></i3>
</i2>
<i2><idxterm>strawberry</idxterm></i2>
</i1>
```

Here is the formatted result:

```
dessert sauces
  butterscotch 12
  hot fudge
    microwave method 12
    stovetop method 12
  strawberry 12
```

You'll notice that the i3 entries, microwave and stovetop methods, are only listed under hot fudge. This is because the i3 tags are listed inside the i2 hot fudge tag. If you wanted the i3 tags to be under butterscotch, hot fudge, and strawberry, you would have to place the i3 tags inside each one of the i2 tags. So you would have microwave method and stovetop method listed three times each in this example. You'll also notice the page numbers are automatically placed in the formatted example. You don't need to print the document then add the page numbers. It's all done for you.

Cross referencing index entries

As you can see from the examples under "Position method" on page 117, repeating the entire structure of primary and secondary entries before each tertiary entry can be pretty tedious. For this reason, IBMIDDoc has the ID, I1ID, and I2ID attributes on the indexing tags to allow you to get at the structure with just the name you put on the ID.

The ID attribute identifies an index entry within an SGML document. IDs must be unique within a single document. IDs can be up to 64 characters long. IDs must start with an alphabetic character and can contain letters, numbers, dashes (-), or periods (.).

When you put an ID attribute on an I1 or I2 tag, the formatter "remembers" *that entry and any higher level entries associated with it*.

For example, if you had these entries:

```
<i1><idxterm>sauces</idxterm>
<i2 id="mayo"><idxterm>mayonnaise</idxterm>
<i3><idxterm>hard way</idxterm></i3></i2></i1>
```

This associates the ID "mayo" with both mayonnaise *and* sauces.

Then you can enter:

```
<i3 i2id="mayo"><idxterm>blender method</idxterm></i3>
 ...
<i3 i2id="mayo"><idxterm>food processor method</idxterm></i3>
```

and get exactly the same results as if you had coded this:

```
<i1><idxterm>sauces</idxterm>
<i2><idxterm>mayonnaise</idxterm>
<i3><idxterm>hard way</idxterm></i3></i2></i1>
 ...
<i1><idxterm>sauces</idxterm>
<i2><idxterm>mayonnaise</idxterm>
```

```
<i3><idxterm>blender method</idxterm></i3></i2></i1>
 ...
<i1><idxterm>sauces</idxterm>
<i2><idxterm>mayonnaise</idxterm>
<i3><idxterm>food processor method</idxterm></i3></i2></i1>
```

When you use the reference attribute on the I2 and I3 tags, they pick up the specified level needed from the structure named with the reference name. (You can't use a reference on an I1 tag, because there are no "higher" levels.)

If you want to pick up *all* the levels (that is, you have an identical structure to the one named with the ID attribute), you should use the IREF tag. Because you are picking up all the levels, the IREF tag doesn't need a level indicator of its own. The IREF tag adds a page number to an existing structure.

So if we had many different ways of making mayonnaise with a blender, we could enter:

```
<i1><idxterm>sauces</idxterm>
<i2 id="mayo"><idxterm>mayonnaise</idxterm></i2></i1>
 ...
<iref refids="mayo">
 ...
<iref refids="mayo">
```

and we would get this result:

```
sauces
   mayonnaise   20, 23, 26
```

You can even use both an ID and a reference on an I2 or I3 tag, to both pick up the higher level entries (the referenced entry) and then give this whole new structure a name (the ID). (You want to be careful not to confuse yourself, though.)

Where to put index entries

Your index entries can be entered just about anywhere; they don't cause any variation in how the text around them is formatted. Here is a list of good places to put index entries to ensure the proper page reference:

- Immediately following a heading, after the ending Dprolog tag.
- Following the first sentence of a paragraph.
- Following the first sentence of a list item or definition description.
- Immediately following an XMP tag or FIG tag (in particular, if you are indexing a figure that is going to float, the index tag must be inside the figure).
- Immediately following the first ENTRY element of the first ROW of a TABLE, for entries that point to an entire table (this is because the page on which the table will begin isn't determined until the first ENTRY element is processed).

For example, because the formatter keeps the first few lines of a paragraph on the same page, using the index tags in these places ensures that the page reference picked up for the entry is the same page on which the paragraph starts. If the index entries were placed before the paragraph, they might be processed (and the page number picked up) before the formatter discovers that it has to start a new page for the paragraph.

It is a good idea to put index entries that don't have page references associated with them in one place in the front of your source document. If you scatter them through your document, you will have trouble finding them when you want to change them because the index itself won’t give you a page number to help you find them.

And where NOT to put index entries: ***Do not*** put index entries in the following places:

- Do not place index entries anywhere before the PREFACE tag or between the BODY tag and the first division in the body. They can cause extraneous blank pages in your BookMaster-formatted document if they occur at either of these points.
- Don’t put index entries in a table between a ROW tag and an ENTRY tag. This is also a Bookmaster-formatting problem. Put them in the Entry tag.
- Don’t put index entries in the middle of a sentence, this raises heck with translation centers. Put them before the paragraph, or before the sentence.

Defining index entries (central indexing)

You can define index entries in the document’s prolog. This is often called “central” or “central-file” indexing.

Use the index definitions (IdxDefs) element to put some or all the index entry definitions in a central place, either the document prolog or a division prolog. This makes it easier for you to maintain your index IDs, since you can maintain them in a central place. Use I1 to define each primary index entry. In IdxDefs, any I2 (secondary) and I3 (tertiary) index entries must be contained in the I1 index entries. The following example shows I1 index entries in IdxDefs for “document structure” and “element”:

```
<idxdefs>
<i1 id="ixdocstruct"><idxterm>document structure</idxterm>
<i1 id="ixelement"><idxterm>element</idxterm></i1>
</idxdefs>
```

No page number is associated with index entries located in the IDXDEFS. Use the IREF element or add lower level index entries to set reference points. For example:

```
<iref refids="ixdocstruct">
<i2 i1id="ixelement"><idxterm>context</idxterm></i2>
```

Creating index entries by cross-indexing

There are many times in indexing when you want to associate the same set of subentries and page references with a group of primary index entries. This process is called “cross-indexing”.

You can use the IdxTerm element multiple times to associate the same set of subentries with several I1-level index terms. For example, to associate the same set of I2 and I3 entries with both “cross indexing” and “indexing, cross”, you can define an I1 index entry as follows:

```
<i1><idxterm>cross indexing</idxterm><idxterm>indexing, cross</idxterm>
<i2><idxterm>creating</idxterm>
<i3><idxterm>easy way</idxterm></i3></i2></i1>
```

You would get this result:

```
cross indexing
    creating
        easy way 12
```

```
 ...
indexing, cross
    creating
        easy way 12
```

In this example, "cross indexing" is the first index term specified and is considered to be the main index term for this entry.

Cross-indexed primaries do not have to have identical subentries. For example, suppose you want to index all custard pies under both "pies" and "custard pies"; all fruit pies under both "pies" and "fruit pies"; and a general discussion of pies under "pies" alone. To do that, use the following markup, the Index definitions are also used and are in the document's prolog.

```
<prolog>
 ...
<idxdefs>
<i1 id="pies"><idxterm>pies</idxterm></i1>
<i1 id="custpies"><idxterm>custard pies</idxterm><idxterm>pies</idxterm></i1>
<i1 id="fruitpies"><idxterm>fruit pies</idxterm><idxterm>pies</idxterm></i1>
</idxdefs>
</prolog>
 ...
<i2 i1id="pies"><idxterm>general discussion</idxterm></i2>
 ...
<i2 i1id="custpies"><idxterm>coconut</idxterm></i2>
 ...
<i2 i1id="custpies"><idxterm>chocolate</idxterm></i2>
 ...
<i2 i1id="fruitpies"><idxterm>peach</idxterm></i2>
 ...
<i2 i1id="fruitpies"><idxterm>blueberry</idxterm></i2>
```

which gives results similar to this:

```
custard pies
  chocolate  2
  coconut  1
 ...
fruit pies
  blueberry  5
  peach  3
 ...
pies
  blueberry  5
  chocolate  2
  coconut  1
  general discussion  1
  peach  3
```

Defining See and See-also references

If you need see and see-also references in your index, use the SeeID and SeeText attributes with I1 or I2 elements. SeeID points to an index entry specified by an ID attribute.

SeeText points to text that you specify. Use SeeID whenever possible because it ensures that you are referring your reader to a real entry in the index. (You see a question mark in your cross-reference listing if a SeeID specifies an ID that does not exist.) This markup shows the SeeID attribute:

```
<i1 id="bech"><idxterm>bechamel sauce</idxterm></i1>
<i1 seeid="bech"><idxterm>white sauce</idxterm></i1>
```

Processing Note: For a see reference to work correctly in XHTML or HTML output, the I1 needs to be in the prolog. Any I1 in the body of the document is treated as an index link, giving you a see-also reference. Also, the ID needs to have an IREF in the body of the document.

IBMIDDoc determines whether the reference should be a see or a see-also reference. If “white sauce” has no page references of its own and no secondary entries, the index reference is a see reference, as follows:

```
bechamel sauce


white sauce
  See bechamel sauce
```

However, if “white sauce” has page references or other subentries, the index reference is a see-also reference, as follows:

```
bechamel sauce 29


white sauce  32
  See also bechamel sauce
```

SeeText works the same as SeeID, except that you supply the text you want for the reference, as follows:

```
<I1 seetext="cakes, cookies, pies"><IDXTERM>desserts</IDXTERM></I1>
```

With SeeText, you must ensure that the referenced entries (in this case, cakes, cookies, and pies) can all be found in your index.

If “desserts” has other references, the index entries appear as follows, with the see-also reference listed first in the subentries:

```
desserts
  See also cakes, cookies, pies
```

If both SeeID and SeeText are specified, only the SeeID is used.

If a SeeID points to the ID of a secondary or tertiary entry, as in the following example, IBMIDDoc constructs the full cross reference for you:

```
<I1><IDXTERM>sauces</IDXTERM><I2 ID="vinaig"><IDXTERM>vinaigrette</IDXTER
   ⋮
<I1 SEEID="vinaig"><IDXTERM>oil and vinegar dressing</IDXTERM>
```

The cross-reference looks similar to this:

```
oil and vinegar dressing
  See sauces, vinaigrette
   ⋮
sauces
  vinaigrette  83
```

When you use SeeID in one index entry to refer to another index entry that has several index terms defined, the “See” or “See also” text generated in the index shows only the main (first) index term.

Note: If a SeeID attribute points to an I1 or I2 element that specifies cross indexing (has multiple IdxTerm elements), the resulting see or see-also reference points only to the first (main) IdxTerm element. Because of that, you should select the main entry carefully for any index elements that specify cross indexing.

For Xyvision generated PDFs, there will not be a PDF link between the “see” entry and the entry it refers to.

Controlling the Index Sorting

This currently does not work.

The automatic sorting of index entries may not always suit your needs, as in these examples:

- When you want to index titles without regard to leading articles; for example, indexing “The Wind in the Willows” under “wind”.
- When you want to index entries that start with a special character according to the first alphabetic character rather than the special character; for example, indexing “&date;” under the Ds instead of the &s.
- When you want numeric subjects to appear in the alphabetic section as if they are spelled out; for example, indexing “8-layer cake” as if it is spelled “eight-layer cake”.

To change the way an index entry is sorted, specify the SortKey attribute. This is currently not supported for the Xyvision PostScript and PDF formatter.

For example, to sort “8-layer cake” as if the “8” is spelled out as “eight”, use the following markup:

```
<i1 sortkey="eight">8-layer cake</i1>
```

This could results in an index like the following:

```
egg substitutes  58
8-layer cake  82
endive, Belgian  75
```

The sort key needs to be only long enough to guarantee that the entry is sorted as you want. For this example, the following markup would be sufficient because ei is enough to ensure the desired sorting.

```
<i1 sortkey="ei">8-layer cake</i1>
```

However, consider making the sort key somewhat longer to ensure the desired sorting when information, such as “8-layer salad”, is added. So you really may want this for your “8–layer cake” sortkey:

```
<i1 sortkey="eight-layer cake">8-layer cake</i1>
```

Generating the index

The INDEX tag shows that you want your index placed in the back matter. In the BACKM (back matter) section, insert an INDEX tag. You should also insert a GENDTITLE tag. It generates the level 1 heading “Index” for you and then includes the sorted and formatted index.

It would look like this:

```
<backm>
<index>
<gendtitle>
</index>
</backm>
```

and that's all there is to that.

You can override the index heading text like so:

```
<index>
<titleblk>
<title>My Cute, Highly Retrievable Index</title>
</titleblk>
</index>
```

To get an idea of what your final index will look like, just look at the index in this book; it was done using these tags.

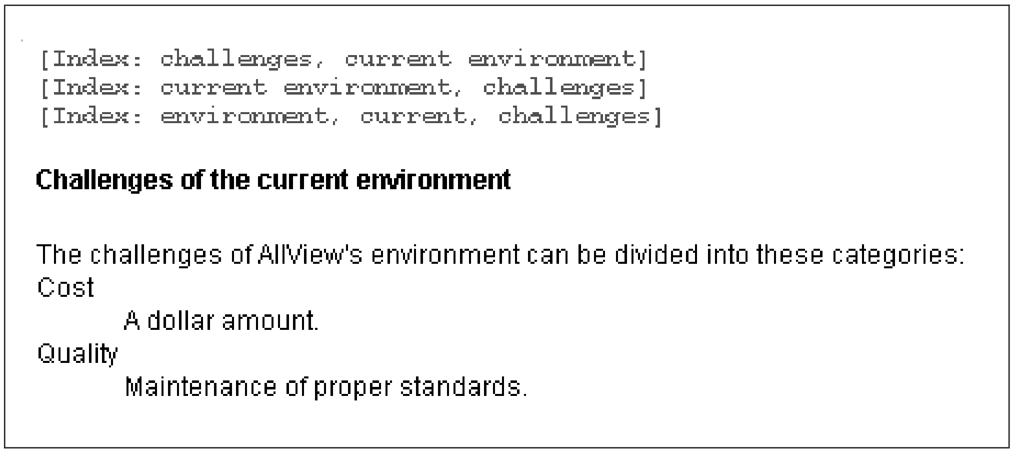
For XHTML and HTML processing, your index entries are added to the meta data for your HTML file. This helps search engines to better find your articles.

Helping online reviewers see your index entries

The XHTML transform has an option to show your index entries in the places they occur. This is currently only for IBMIDDoc to XHTML processing; to help with online reviews. You specify this when you transform your IBMIDDoc to XHTML; the XHTML-2 page has an option named **Display index entries in document body**(/INDEXSHOW). Given this source:

```
<d id="challenge">
<dprolog><titleblk>
<title>Challenges of the current environment</title>
</titleblk></dprolog>
<i1><idxterm>challenges, current environment</idxterm></i1>
<i1><idxterm>current environment, challenges</idxterm></i1>
<i1><idxterm>environment, current, challenges</idxterm></i1>
<dbody>
<p>The challenges of Allview's environment can be divided into these categories:
<dl>
<dlentry><term>Cost</term>
<defn>A dollar amount.</defn></dlentry>
<dlentry><term>Quality</term>
<defn>Maintenance of proper standards.</defn>
</dlentry>
</dl></p>
</dbody></d>
```

This is what you'll see on the corresponding XHTML web page; the index entries are colored and appear as follows:


[Index: challenges, current environment]
[Index: current environment, challenges]
[Index: environment, current, challenges]

Challenges of the current environment

The challenges of AllView's environment can be divided into these categories:

Cost
 A dollar amount.

Quality
 Maintenance of proper standards.

Creating a master index

A master index incorporates the index entries from other documents and combines them into one central place for the user The master index provides the name of the document and the page number where the information about the index entry can be found.

To create a master index for a set of documents, do the following:

1. Each of the documents that contribute to a master index needs to have the master index prefix specified in their prolog. In each contributing book, use a MasterIndexInfo element containing a MasterIndexPrefix element to specify the prefix code. For example, for a user guide, you might want to use the prefix USERGD; for a reference, you might want to use the prefix REF. For example:

   ```
   <masterindexinfo>
   <masterindexprefix>USERGD</masterindexprefix>
   </masterindexinfo>
   ```

   The prefix should be something short, generally less than 10 characters. We also recommend having no spaces. When an entry in the master index prints, they will look something like the following. This sample master index has three books: USERGD for a user's guide, INTRO for an introduction, and PLAN for a planning guide. The page number after the prefix is the page number in the corresponding book.

   configuring INTRO-2, USERGD-12
   changing PLAN-34

   deleting USERGD-39
2. To format a master index using Xyvision:
   a. This step is optional. The master index support allows you to link directly from a PDF of the master index to that page in the PDF of the contributing document. To do this, you can specify an ExternalFileName element in each contributing document to be contained in the master index. Specify only the file name of the document. Do not specify a file extension. This only works for Xyvision-formatted documents. For example, this specifies the name of this document is myusergd. If you will be placing the PDF files on an AIX® server, remember that the file names are case-sensitive.

      ```
      <externalfilename>myusergd</externalfilename>
      ```
   b. Each of the documents to be contained in the master index needs to be formatted for PostScript using Xyvision. By specifying the master index

prefix elements, the Xyvision formatter generates a PostScript file and a master index file (file extension MDX) for each document. The document must also have an Index tag, and be processed so that an index is generated (avoid the NOINDEX option).

c. Once all of the contributing documents have been formatted and the master index files (MDX) have been created, you create a master index document that imbeds each of the individual master index files. You indicate that this is a master index document by coding a MasterIndex element containing a MasterIndexObj element for each MDX file to be included. Each MDX file needs to be declared; declare the MDX file as a ″graphic″ entity with a notation of ″mindex″.

d. Format the master index file for PostScript using Xyvision to create the master index document.

e. If you want, create Adobe Acrobat PDFs from the PostScript files for the master index and the contributing documents.

3. To format a master index using BookMaster:

   a. Each of the documents to be contained in the master index needs to be formatted for PostScript using BookMaster. You will need to format the documents without using the ID Workbench. The master index files from BookMaster processing are not returned to the OS/2® IDWB client.

   b. Transform each contributing document to BookMaster.

   c. Transform the master index document to BookMaster.

   d. Upload each converted document, with its artwork, to VM and process them using IDPS. For each contributing book, specify the correct BookMaster master index options for IDPS to create the master index; either:

      ```
      Master index ==> filename
      ```

      or:

      ```
      SYSVAR (M filename)
      ```

      where *filename* is the name for the contributing document's master index file. This creates the file: *filename* DSMMINDX.

   e. For each contributing document, you will need to add an imbed command for the master index file in the following format. Add these lines just before the INDEX tag in the master index document.

      ```
      .* set the name of the DSMMINDX file
      .namefile name=filename cms='filename dsmmindx'
      .* imbeds the index source
      .im filename
      ```

This shows an example prolog for a contributing document. The ExternalFileName specifies the file name of a document: idfgsmst, without the file extension. The MasterIndexInfo and MasterIndexPrefix elements indicate the prefix is GSUG (the prefix used for this book).

```
<ibmbibentry><doctitle>
<titleblk>
<title>Getting Started and User's Guide</title>
</titleblk></doctitle>
<externalfilename>idfgsmst</externalfilename>
</ibmbibentry>
<masterindexinfo>
<masterindexprefix>GSUG</masterindexprefix>
</masterindexinfo>
```

In the master index document, the contributing MDX master index files must be declared. This example shows two sets of declares. The ″mindex″ declares are for the master index files; the ″sgmldoc″ declares are for cross-book links (using the citations).

```
<!ENTITY instidx SYSTEM "idfinmst.mdx" ndata mindex>
<!ENTITY planidx SYSTEM "idfplmst.mdx" ndata mindex>
<!ENTITY gsugidx SYSTEM "idfgsmst.mdx" ndata mindex>
<!ENTITY inst SYSTEM "idfinmst.idd" ndata sgmldoc>
<!ENTITY plan SYSTEM "idfplmst.idd" ndata sgmldoc>
<!ENTITY gsug SYSTEM "idfgsmst.idd" ndata sgmldoc>
```

This shows a sample master index document:

```
<ibmiddoc>
<prolog><ibmbibentry><doctitle>
<titleblk>
<title>Master Index</title>
</titleblk></doctitle>
</ibmbibentry>
<bibentrydefs>
<ibmbibentry docname="gsug" id="gsug"><doctitle><titleblk><title>
Getting Started and User's Guide</title></titleblk></doctitle>
</ibmbibentry>
<ibmbibentry docname="inst" id="inst"><doctitle><titleblk><title>
Workstation Installation Guide</title></titleblk></doctitle>
</ibmbibentry>
<ibmbibentry docname="plan" id="plan"><doctitle><titleblk><title>
Planning and Host Installation Guide</title></titleblk></doctitle>
</ibmbibentry></bibentrydefs>
</prolog>
<frontm style="display='cover'">
<toc><gendtitle></toc>
</frontm>
<body>
<d>
<dprolog><titleblk>
<title>Master Index Prefix Codes</title>
</titleblk></dprolog>
<dbody>
<dl>
<dlentry><term>GSUG</term>
<defn><cit bibid="gsug"></defn>
</dlentry>
<dlentry><term>INST</term>
<defn><cit bibid="inst"></defn>
</dlentry>
<dlentry><term>PLAN</term>
<defn><cit bibid="plan"></defn>
</dlentry>
</dl>
</dbody></d>
</body>
<backm>
<masterindex>
<specdprolog><gendtitle></specdprolog>
<masterindexobj obj="gsugidx">
<masterindexobj obj="planidx">
<masterindexobj obj="instidx">
</masterindex></backm>
</ibmiddoc>
```

Chapter 12. All about linking

Hypertext links (we'll just call them links from now on) connect elements in one part of an online document to elements in another part of the same document or a separate online document.

Linking 101

Think of links as you would think of cross references in a printed document. For example, while reading about Henry David Thoreau in the encyclopedia, a reader comes across a reference to another topic: "See also *Ralph Waldo Emerson*". What does the reader do? Keeps a finger on the page that describes Thoreau and turns back to the new reference. The reader has just created a link from one part of the document to another.

In printed documents, a reader turns to related information. In online documents, IBMIDDoc creates a link to related information, and the online reader can then display that information. The way a reader selects a reference, that is, asks the online browser to display it, is usually by double-clicking on some highlighed text. Web browsers, Adobe Acrobat, and BookManager Read are the typical online browsers for which we create books or articles.

The following terms are used for the different types of links:

Cross-reference links
These are explicit links that IBMIDDoc creates between cross references that use the XREF tag and referenced information within one document.

Author-defined links
These are explicit links that you specify with the L (link) tag and others. These can be within a document or from one document to another.

Associative links
These are links that BookManager creates automatically. You do not need to specify them. They typically come from glossary terms

Implicit links
These links are derived from the structure of the markup. Tables of contents and index entries are examples of implicit links derived from the SGML markup structure.

| It may be obvious, but please do not forget to test your links. There is also
| maintenance between releases in checking links, as things you link to might move
| to other locations on the web.

Creating links within a document

Generic links support the identification of a "hot spot" as one anchor of the link, and the specification of a target as the other end of a link. Within the same document, you can use the XREF tag or the L tag to create links. The XREF tag uses the heading text or figure number, for example, as the "hot spot" text. If you want to use your own text for the link, perhaps for readability or to have the link fit better in a sentence, use the L (link) element. The content of the L element is the "hot spot" text, and the linkend attribute specifies the ID of the other anchor.

This next example shows how to make both an XREF tag and an L tag link to a heading. The L tag's LinkEnd attribute references the division's ID attribute.

```
<d id="xrefhyl">
<dprolog><titleblk>
<title>All about linking</title>
</titleblk></dprolog>
<dbody>
<p>Hypertext links (we'll just call them links from
now on) connect elements in one part of an online
document to elements in another part of the same document
or a separate online document. </p>
 ...
<p>Sometimes you need to <l linkend="xrefhyl">link</l> to
other topics.</p>
 ...
<p>See <xref refid="xrefhyl"> for ways of creating links.</p>
```

The XREF appears as a cross reference with a page number in a hardcopy document. In an online document, the heading text "All about linking" is highlighted and selectable. The L type of link has no representation at all in a hardcopy document. But, in an HTML, PDF, or BookManager online version, the text "link" is highlighted and selectable. When you select the link, the browser jumps to the division. Why the difference you ask? There are times when you want to control the text in link; for this, you use the L tag.

Here's how they appear:

Formatted Example

Sometimes you need to link to other topics.

See Chapter 12, "All about linking" on page 129 for ways of creating links.

End of Formatted Example

You can link to other elements in the same document using this same linking mechanism. The target of an explicit link should be to an ID on the outer container (for example, D, MSG, LE, FIG, TABLE) and not the title text or caption text.

If you want to link to something that does not have a title, caption, or other generated text, you still have a choice.

- You can use an XREF tag to point to another element (like a P tag), and on that target element, use the XREFTEXT attribute to specify the linking text. This allows you to cross-reference, get your page number for hardcopy or PDF, and still have a link.
- Use an L tag to point to the other element; the content of the L tag contains the linking text.

Here's a sample of a link and a cross-reference to a paragraph:

```
<p id="paraxref" xreftext="cute, little paragraph">
Here's a cute, little paragraph that I want to link
to. It has no caption so I need to add the XREFTEXT
attribute.</p>
<p>See the <l linkend="xrefhyl">cute paragraph</l> for another way to link.</p>
<p>See <xref refid="paraxref"> for ways of creating links.</p>
```

Here's how it comes out:

Formatted Example

Here's a cute, little paragraph that I want to link to. It has no caption so I need to add the XREFTEXT attribute.

See the cute paragraph for another way to link.

See "cute, little paragraph" for ways of creating links.

End of Formatted Example

Linking to another document

There are several classes of inter-document links to other documents:

- Linking to another IBMIDDoc document.

  This type of link is interpreted based on the output being produced for the linking (from) document. The link produced in the output of the linking document assumes the same type of processing is done for the target document. So Xyvision documents produced from IBMIDDoc will link to the Xyvision output of the target document, HTML documents to other HTML documents, IPF documents to other IPF documents, and so forth.
- Linking to a specific output type of document.

  This type of link specifies the type of document to be linked (for example, HTML or IPF). The type of output processing done to the linking document does not affect the type of the target, which remains the same.

Note: If the ID Workbench output transform application finds an ID that conforms to BookMaster's ID rules (seven characters or less, no special characters, starting with an alphabetic character), it will preserve the ID when it transforms the SGML markup to BookMaster markup. This enables both cross-document links using BookManager and cross-document references between IBMIDDoc documents, and between IBMIDDoc documents and native BookMaster documents.

Citation link to an IBMIDDoc document

You use the CIT element to reference another document as a whole. If the bibliographic entry specifies an entity declaration for the bibliographic entry and uses the DocName attribute, a link to that other document is created. For BookManager to use this link properly, specification of IBMDocNum is also necessary.

This markup generates the appropriate cross-document link markup in Xyvision PDFs and BookManager.

```
<!ENTITY bk2ent SYSTEM "xdoclnk2.idd" NDATA sgmldoc>
 ...
<ibmbibentry docname="bk2ent" id="bk2">
<doctitle><titleblk><title>Target Document (XDOCLNK2)</title>
</titlblk></doctitle>
<ibmdocnum>SC41-0002</ibmdocnum>
</ibmbibentry>
 ...
<p>Title citation link: See the <cit bibid="bk2"> for this
information.<p>
```

To implement this, you need to do the following:

1. Declare the target book as a “graphic” entity. The name `bk2ent` is the name that is used in the IBMBibEntry DOCNAME attribute. The system ID `xdoclnk2.idd` needs to be the target name of the book you are linking to; typically this is a PDF or HTML file. The file name needs to match the target PDF or HTML file name. The notation data is an `sgmldoc`.
2. Create an IBMBibEntry for the book. The DOCNAME attribute points to the declared book name. The ID you assign is used on Cit tags and NameLoc/NMList tags.

Linking to an XHTML, HTML, or web document

A link to an XHTML or HTML document (or location within a web document) is accomplished by referencing its URL. This is done by referencing, by ID, a notation location or NOTLOC element with a specified notation of URL which contains the URL.

Here is an example that will link to the main IBM web page.

```
<ldescs>
<notloc id="ibm" notation="url">http://www.ibm.com</notloc>
<ldescs>
</prolog>
 ...
<p>You should try linking to the
<l linkend="ibm">IBM home page</l>.</p>
```

Formatted Example

You should try linking to the IBM home page.

End of Formatted Example

When processing for HTML or Xyvision PDF output, the appropriate anchor markup is generated. When processing for other outputs, the URL is ignored.

Here is another example that will links to a PDF version of a book:

```
<ldescs>
<notloc id="gsugpdf" notation="url">
http://w3.rchland.ibm.com/projects/IDWB/documents/idfgsmst.pdf</notloc>
<ldescs>
</prolog>
 ...
<p>You should try linking to the
<l linkend="gsugpdf">PDF version of the IDWB Getting Started book</l>.</p>
```

Formatted Example

You should try linking to the PDF version of the IDWB Getting Started book.

End of Formatted Example

Linking to items in another IBMIDDoc document

We showed you in "Citation link to an IBMIDDoc document" on page 131 how to link to a Xyvision PDF document as a whole. How would you like to link to a specific heading, figure, or table within a Xyvision PDF book? Here's how! This also works for BookManager cross-book links (from one BookManager book to another).

You can also link from one XHTML or HTML "book" to another (from one set of XHTML or HTML files to another set of XHTML or HTML files). For this to work, you need to format the XHTML or HTML files; and save the resulting IDX file from the IBMIDDoc to XHTML or IBMIDDoc to HTML process.

You find the ID of that heading, figure, or table on the target book, and set up your LDesc and NameLoc tags to point to those IDs. Then you make Links to those NameLoc definitions, and Xyvision and Acrobat do the rest.

For example, you have the following things you want to reference in a book named "fred.pdf":

- Heading ID: barney
- Figure ID: betty
- Table ID: wilma

The declaration for the SGML document must have the name matching the PDF name; the extension can be IDD. The NameLoc tags set up the links that are used later:

```
<!ENTITY fred SYSTEM "fred.idd" NDATA sgmldoc>
 ...
<ldescs>
<nameloc id="barneyintro" objtype="head">
<nmlist docname="fred">barney</nmlist>
</nameloc>
<nameloc id="bettyphoto" objtype="fig">
<nmlist docname="fred">betty</nmlist>
</nameloc>
<nameloc id="wilmachart" objtype="table">
<nmlist docname="fred">wilma</nmlist>
</nameloc>
</ldescs>
 ...
<p>Barney's hobbies are listed <l linkend="barneyintro">here</l>.
This is Barney's wife, <l linkend="bettyphoto">Betty</l>.
Wilma divides her time <l linkend="wilmachart">this way</l>.</p>
```

Sometimes you need to use the softcopy book name or some other name for the PDFs. If this is the case; you will need to ensure the declarations for the targeted books have the system ID set as the final file name of that target book.

The *ID Workbench Getting Started and User's Guide* has more information about the processing needed for this type of linking; see:

- Linking in Xyvision-Formatted Books
- XHTML and HTML linking concepts

Making a graphic a link

To make a graphic a link, you use the MMObjLink tag within the MMObj tag. Sometimes you want a picture to link to an article. For example, this graphic links to the main topic in this section:

Here’s how you code that:

```
<mmobj placement="inline"><objref obj="tocdoc">
<mmobjlink linkend="xrefhyl"></mmobjlink>
<textalt>table of contents icon</textalt></mmobj>
```

The MMObjLink tag can contain an AreaDef tag; AreaDef is currently not supported in the output transforms. The entire graphic becomes the link.

By using a 1x1 pixel graphic, you can make what is called a “skip link” that you can use to have a screen reader bypass something. Only the screen reader will read the alternative text; and the link will be easy for the reader to select. The sighted user will not encounter the link. Here’s how you might use a skip link:

```
►►──COMMAND-NAME──THIS=that-value──────────────────────────────────────►◄
```

Here’s text that follows the diagram.

Here’s how that was coded:

```
<p><mmobj><objref obj="hidden">
<mmobjlink linkend="idafterdiag"></mmobjlink>
<textalt>Skip reading of syntax diagram.</textalt>
</mmobj><syntax>
<group>
<kwd>COMMAND-NAME</kwd>
</group>
<group choiceseq="composite"><kwd>THIS</kwd><delim>
=</delim><var>that-value</var></group>
</syntax></p>
<p id="idafterdiag">Here's text that
follows the diagram.</p>
```

Linking to an IPF document

A direct link to an IPF book may be coded similar to this example:

```
<!ENTITY sctagent SYSTEM "SCTAGENT.INF" NDATA IPFINF>
 ...
<nameloc id="ID907" objtype=book>
<nmlist nametype=entity>sctagent</nmlist>
</nameloc>
 ...
<p>This paragraph links to an IPF online book.
See this <l linkend=ID907 style="IPF: (data='sctagent.inf'
object='view.exe' reftype=launch)">IPF topic</l> for more info.</p>
```

In the example, the NAMELOC only defines the ID referenced by the link. The entity declaration performs no function at all. This coding reflects the coding that should be used in the future when the need for the passthrough attributes has been eliminated.

This coding will generate the appropriate IPF code but does not produce usable Xyvisoin PDF, BookManager, or HTML code. Cross document (XREF) references are not supported in IPF.

For IPF, the citation element alone does not generate a cross-document link. The link must be coded with the appropriate IPF passthrough attribution to generate a launch-type link, launching the IPF viewer against the desired IPF file.

```
<!ENTITY bk2ent SYSTEM "xdoclnk2.idd" NDATA sgmldoc>
 ...
<nameloc id="lkl" objtype="BOOK">
 <nmlist nametype=entity>bk2ent</nmlist>
 </nameloc>
 ...
<ibmbibentry docname="bk2ent" id="bk2">
 <doctitle><titleblk><title>Target document (XDOCLNK2) </title>
 </titleblk></doctitle>
 <ibmdocnum>SC41-0002</ibmdocnum>
 </ibmbibentry>
 ...
 <p>Title citation link: See the <cit bibid="bk2" props="#not IPF">
 <l linkend=lkl props="IPF" style="ipf:(data='xdoclnk2.inf'
reftype='launch' object='view.exe')">
 Target document (XDOCLNK2)
 </l>
 for this information.</p>
```

Chapter 13. Glossaries

Glossaries are similar to definition lists, in that you pair terms with their definitions, using the Term and Defn (definition) elements. You begin your glossary with the Glossary element, which generates a head level 1 with the generic heading “Glossary.” The Glossary element, can have an ID attribute for cross-referencing. You can use your own title instead of “Glossary,” so you might have ″Definition of terms″.

The glossary typically goes in the back matter, in the Backm element’s content. Here are the typical tags for the glossary section:

```
<backm>
 ...
<glossary>
<specdprolog><gendtitle></specdprolog>
<dbody>
<gl>...</gl>
</dbody>
</glossary>
 ...
</backm>
```

If you like, you can enter ordinary text after the DBody before you actually begin your glossary list with its entries. You start the glossary list with a GL tag and end it with its matching end tag. Within the glossary list, you use the GlEntry, Term, and Defn tags to mark up the terms and their descriptions. The description can be many paragraphs. For example, the glossary in this book was entered in part like this:

binding edge. The edge of a page to be bound, stapled, or drilled.

Here’s its markup:

```
<gl>
<glentry><term>binding edge</term>
<defn>The edge of a page to be bound, stapled, or drilled.
</defn>
</glentry>
</gl>
```

If your term has multiple definitions, just enter another set of Defn elements in the Glentry. For example:

cat. (1) cute, furry mammal that purrs when rubbed the right way (2) owner of the house in which it dwells, any people sharing the dwelling are the caretakers

Here’s its markup:

```
<gl>
<glentry><term>cat</term>
<defn>cute, furry mammal that purrs when rubbed the
right way</defn>
<defn>owner of the house in which it dwells, any people
sharing the dwelling are the caretakers</defn>
</glentry>
</gl>
```

Defining Terms

Use the GLEntry element to define a term used in your document. GLEntry contains the glossary term and one or more Defn elements, each of which contains a definition for the term. You can define terms in the document prolog or in a glossary list.

Glossary entries for IPF output become divisions displayed in popup windows.

In your document, you can use a Termdef attribute on a Term element that points to the ID of the glossary term. In HTML, IPF, and Window's Help, this generates a link from the Term element to the term in the glossary.

Separating letter groups in a glossary

The retrievability of items in your glossary will be improved if you use the GLBlk (glossary block) elements for alphabetic groups of terms. For example:

A

aardvark. long-nosed doglike creature.

B

bat. flying mouse

Here's its markup:

```
<gl>
 <glblk><title>A</title>
  <glentry><term>aardvark</term>
  <defn>long-nosed doglike creature.</defn>
  </glentry>
 </glblk>
 <glblk><title>B</title>
  <glentry><term>bat</term>
  <defn>flying mouse</defn>
  </glentry>
 </glblk>
</gl>
```

Defining Classes for Terms

You can also define classes of glossary terms, and assign properties to those classes, as shown in the following example, where the terms are defined as being in either class *odwords* or class *duckwords*:

```
   ⋮
<CLASSDEF ELETYPES="GLENTRY" CLASSNAME="odwords">
 <TITLE>OTHER D-WORDS</TITLE>
  <SEM>OTHER WORDS BEGINNING WITH A D</SEM>
</CLASSDEF>
<CLASSDEF ELETYPES="GLENTRY" CLASSNAME="duckwords">
 <TITLE>OTHER D-WORDS</TITLE>
  <SEM>WORDS ABOUT DUCKS</SEM>
</CLASSDEF>
   ⋮
<GL>
 <GLENTRY CLASS="odwords"><TERM>December</TERM>
  <DEFN>A month that is often cold and dreary.</DEFN></GLENTRY>
 <GLENTRY CLASS="odwords"><TERM>duty</TERM>
```

```
  <DEFN>What one must do in life.</DEFN></GLENTRY>
 <GLENTRY CLASS="duckwords"><TERM>ducks and drakes</TERM>
  <DEFN>The game of skimming stones across water.</DEFN></GLENTRY>
 <GLENTRY CLASS="duckwords"><TERM>ducky</TERM>
  <DEFN>Very well, as in <q>Just ducky, thanks.</q></DEFN></GLENTRY>
 <GLENTRY CLASS="odwords"><TERM>Durango</TERM>
  <DEFN>City in Colorado.</DEFN>
  <DEFN>City somewhere else.</DEFN></GLENTRY>
</GL>
```

For more information about defining classes, see Chapter 20, “Property and Class Definitions” on page 201.

Chapter 14. Bibliographies and citations

This section introduces the IBMIDDoc bibliographic elements. These elements identify books and documents, and allow you to create citation references (and links) as well as traditional “back of the book” bibliographies.

IBMIDDoc has two sets of bibliographic elements.
- The BibEntry elements contain non-IBM bibliography information
- The IBMBibEntry elements contain IBM bibliographic information. IBMBibEntry elements should be used to describe all IBM documents.

In most respects, IBM and non-IBM bibliographic elements are the same. Unless otherwise noted, the information contained in this chapter which refers to a non-IBM-specific bibliographic element (for example BibEntry) also applies to both the IBM-specific bibliographic element (IBMBibEntry).

These elements use the bibliographic entries contained in the BibEntryDefs element by referring to the individual BibEntry’s ID.
- Cit — title citation
- BibList — bibliography list
- LibEntry — library entry

Identifying books and documents

You identify books and documents as bibliographic items using BibEntryDefs elements. The BibEntryDefs element can be used in a Prolog (for your whole document to use) or in a DProlog (for that division to use). BibEntryDefs contain one or more BibEntry elements, which contain individual bibliographic entries.

You can set up a file entity to contain a library of BibEntry elements, and then imbed that file in the BibEntryDefs. This is useful when all the books in your library use the same bibliographic information to create bibliographies.

Each BibEntry (or IBMBibEntry) element can contain extensive bibliographic information about a publication. Each BibEntry must contain a DocTitle element. It can also contain Author, Desc, Publisher, PrtLoc, DocNum, PartNum, ISBN, and PubID. An IBMBibEntry element can contain the same elements, plus two other elements, IBMDocNum and IBMPartNum, which contain IBM-specific publication information.

Here is the markup for defining two books. The IDs can be used to generate citations and bibliographies. The DOCNAME attributes point to an external entity that declares a book to cross-reference to, using a CIT (citation) tag.

```
<bibentrydefs>
<ibmbibentry docname="fruitbats" id="fruitybat"><doctitle>
<titleblk><title>The Care and Feeding of Fruit Bats
</title></titleblk></doctitle>
<ibmdocnum>ZZ99-9876-00</ibmdocnum>
</ibmbibentry>
<ibmbibentry docname="vampbats" id="vampbat">
<doctitle><titleblk><title>The Vampire Bat, a much
```

```
maligned creature</title></titleblk></doctitle>
<ibmdocnum>ZZ99-1234-00</ibmdocnum>
</ibmbibentry>
</bibentrydefs>
```

Here are the declarations for the two books:

```
<!ENTITY vampbats SYSTEM "vampbats.idd" ndata sgmldoc>
<!ENTITY fruitbats SYSTEM "fruitbats.idd" ndata sgmldoc>
```

Using title citations

The Cit element represents a citation of another document. The Cit can either refer to a BibEntry or LibEntry by ID, or include a BibEntry or LibEntry element. The example that follows is a Cit that references the books defined in "Identifying books and documents" on page 141:

> See this book *The Care and Feeding of Fruit Bats* and that book *The Vampire Bat, a much maligned creature*, ZZ99-1234-00 for serious bedtime reading.

Here's its markup:

```
See this book <cit bibid="fruitybat"> and that book
<cit bibid="vampbat" form="full"> for serious bedtime reading.
```

Here's an example citation that is self-contained:

> See these books for a good read and then a weird read: *Tom Sawyer* and *System/36: Concepts and Programmer's Guide*

Here's its markup:

```
See these books for a good read and then a weird read: <cit>
<bibentry><doctitle><titleblk><title>Tom Sawyer</title>
</titleblk></doctitle></bibentry></cit> and <cit>
<ibmbibentry><doctitle>
<library><titleblk><title>System/36</title></titleblk>
</library>
<titleblk><title>Concepts and Programmer's Guide</title>
</titleblk></doctitle></ibmbibentry></cit>
```

The default document style determines the appearance, or form, of the citation. You can specify the form of the Cit by using the FORM attribute. This allows you to specify that only the title or document number will be displayed. You can also use the FORM=FULL specification to cause the entire bibliographic entry to be displayed.

When LibEntry is specified in a Cit element, the LibEntry is collected for use in generated bibliography.

Citations

When the Cit element is used in IBMIDDoc, the link to the target is automatically generated at processing time. Citations must use bibliographic entries to define the target of the citation. If the bibliographic entry specifies an entity using the DOCNAME attribute, the citation may also be treated as a link as well as a citation by the document name of the target. All targets must be defined in a BibEntryDefs element in a Prolog, DProlog, or SpecDProlog element. A central file containing a master BibEntryDefs element with all of the IBMBibEntry and BibEntry elements for a product library can be referenced using an entity reference in your document.

The Cit element uses the BIBID attribute to reference the ID value of the target citation reference that is defined in the IBMBibEntry or BibEntry element contained in a BibEntryDefs element. The example that follows illustrates how to use these elements.

```
<!ENTITY fredbook SYSTEM "fred.idd" ndata sgmldoc>
 ...
<bibentrydefs><ibmbibentry docname="fredbook" id="fred">
<doctitle><titleblk><title>Phred's Guide to Phishing
</title></titleblk></doctitle></ibmbibentry></bibentrydefs>
 ...
<p>See <cit bibid="fred"> for most excelent tips on
catching walleyes.</p>
```

Generating a bibliography

In most cases, bibliographic references are listed in individual BibEntry elements that are contained in the BibEntryDefs element in the Prolog element. Each of these bibliographic references usually has an ID attribute. This ID allows the BibEntry to be referred to in Cit and LibEntry elements. The LibEntry element contains the IDs of the BibEntry elements that make up that library.

To create the markup for a bibliography, you create a BIBLOG section in the back-matter. Then, enter citation tags inside an unordered list. For example:

```
<bibliog><specdprolog><gendtitle></specdprolog><dbody>
<ul> <li><cit bibid="fruitybat" form="full">, describes everything about fruit bats.</li>
<li><cit bibid="vampbat" form="full">, describes everything about vampire bats.</li>
</ul> </dbody></bibliog>
```

Automatic Bibliographies

Wish this was true — but it is just not supported.

When bibliographic elements are arranged as described in the preceding paragraph, a Bibliography will be generated when the SPEC attribute value is AUTO.

```
<BIBLIOG>
<P>A list of the documents referred to in this book....
follows.
<BIBLIST SPEC="AUTO" FORM="full"><GENDTITLE>
```

Defining library entries

LibEntry and IBMLibEntry elements are used to structure and organize information about libraries and collections of documents. You can use IBMLibEntry elements within IBMBibEntryDefs (or BibEntryDefs) , BibList, and Cit elements. The LibEntry element performs the same function as an IBMLibEntry element, but applies only to non-IBM documents.

IBMLibEntry contains the Title of the library. IBMLibEntry can also contain Publisher, PrtLoc, IBMBofNum, IBMPartNum, Prod, ISBN, PubID, ContainedDocs, and Desc elements. IBMLibEntry indicates which books are in the library it describes by referencing the IBMBibEntry elements that describe them. It can contain a list of individual IBMBibEntry elements, or it can contain elements and links that refer to IBMBibEntry elements contained in BibEntryDefs. These entries are referenced using the CONTAINEDDOCS attribute.

The IBMLibEntry in the example that follows shows the ContainedDocs element that references two books:

```
<bibentrydefs>
<ibmlibentry>
<library><titleblk><title>BS/300</title></titleblk>
</library>
<ibmbofnum>SBOF-1234-0</ibmbofnum>
<containeddocs bibids="booka bookb"></ibmlibentry>
<ibmbibentry id="booka"><doctitle><titleblk><title>
BS/300 Guide</title></titleblk></doctitle></ibmbibentry>
<ibmbibentry id="bookb"><doctitle><titleblk><title>
BS/300 Reference</title></titleblk></doctitle></ibmbibentry>
<libentry>
<library><titleblk><title>Back'n'Recovery</title>
</titleblk></library>
</libentry>
</bibentrydefs>
```

Linking BibEntry elements and other documents

A BibEntry and all references to it are links to the document the BibEntry describes. Using the DOCNAME attribute on the BibEntry element allows you to refer to an SGML entity that represents the document being described in the BibEntry. When this attribute is used, any element that refers to the BibEntry will also become a link to the document represented by the SGML entity referred to by this DOCNAME attribute.

An example of using BibEntry and BibEntryDefs

The example that follows illustrates a common usage of the BibEntryDefs and BibEntry elements.

```
   ⋮
<PROLOG>
   ⋮
<LDESCS>
 <NAMELOC ID="UGNAME" OBJTYPE="BOOK">
  <NMLIST DOCNAME="UGX">USERGIDE</NMLIST>
 </NAMELOC>
</LDESCS>
<BIBENTRYDEFS>
 <IBMBIBENTRY ID="BOOK1">
  <DOCTITLE>
   <TITLEBLK><TITLE>IBMIDDOC MIGRATION GUIDE</TITLE>
   </TITLEBLK>
  </DOCTITLE>
 </IBMBIBENTRY>
 <IBMBIBENTRY ID="BOOK2">
  <DOCTITLE><TITLEBLK><TITLE>IBMIDDOC REFERENCE</TITLE></TITLEBLK>
  </DOCTITLE>
 </IBMBIBENTRY>
 <IBMBIBENTRY DOCNAME="UGX" ID="BOOK3">
  <DOCTITLE>
   <LIBRARY><TITLEBLK><TITLE>IBMIDDOC</TITLE></TITLEBLK></LIBRARY>
   <TITLEBLK><TITLE>IBMIDDOC USER'S GUIDE</TITLE></TITLEBLK>
  </DOCTITLE>
  <AUTHORS><AUTHOR><PERSON>
   <NAME>Fred Mertz</NAME>
   <ADDRESS>
    <INTERNET>fredmd@usa.ibm.com</INTERNET>
    <PHONE>212-555-4062</PHONE>
   </ADDRESS>
  </PERSON></AUTHOR>
 </AUTHORS>
 <PUBLISHER>
  <CORPNAME>IBM CORPORATION</CORPNAME>
 </PUBLISHER>
 <IBMDOCNUM>SH21-0783-01</IBMDOCNUM>
</IBMBIBENTRY>
<IBMLIBENTRY>
 <LIBRARY ID="IDDOCLIB">
  <TITLEBLK><TITLE>IBMIDDOC LIBRARY</TITLE></TITLEBLK>
 </LIBRARY>
 <CONTAINEDDOCS BIBIDS="BOOK1 BOOK2 BOOK3"></IBMLIBENTRY>
 </BIBENTRYDEFS>
</PROLOG>
<BODY>
<D>
   ⋮
<P>FOR MORE INFORMATION, SEE <XREF REFID="UGNAME" OBJTYPE="BOOK"></P>
</DBODY>
</D>
</body>
   ⋮
```

Chapter 15. Programming Syntax Diagrams

IBMIDDoc contains a number of elements that are used to define program syntax diagrams. These elements include:

- Syntax; contains the diagram and the markup.
- Delim; delimiters, such as commas and parentheses.
- Fragment; a portion of the diagram.
- FragRef; a reference to a fragment.
- Group; gathers parts of the diagram together.
- Kwd; a keyword, such as something that must be entered or chosen.
- Oper; an operator, such as a plus sign.
- RepSep; a way of repeating and specifying a separator for the repeat.
- Sep; a separator.
- SynBlk; combines groups together.
- SynNote; a diagram footnote.
- SynPh; a syntax phrase.
- Var; a variable, such as a file name.

This chapter contains some general information about creating syntax definitions, and examples of the IBMIDDoc markup are used to obtain the formatted output.

Migration Note

If you are familiar with Bookmaster syntax definitions, you will notice several differences when using IBMIDDoc syntax definitions:

- RepSep definitions
- Descriptions (now within the group or fragment)
- Group and Syntax elements can have titles

For conversion purposes, the outermost group element, which contains all of the other groups in a typical syntax definition, may need to be broken up into several groups, in order to accommodate BookMaster conversion limitations.

Defining the syntax diagram

IBMIDDoc provides elements and attributes that let you create program syntax diagrams. A sample syntax diagram is shown below:

SAA CPI Database Reference

```
►►──ERASE──┬─FORM──┬──name──┬──────────────────────────┬──────────────────────►◄
           ├─PROC──┤        └─(CONFIRM=──┬─YES─┬───────┘
           ├─QUERY─┤                     └─NO──┘
           └─TABLE─┘
```

The sample diagram includes the following syntax diagram elements:

Syntax
: The diagram itself. In the sample diagram, the diagram is set off from the text by a labeled box and contains the diagram title, "SAA CPI Database Reference." IBMIDDoc provides the Syntax element and its end element to define a syntax diagram. The Syntax element has attributes that let you specify the characteristics of the diagram.

Groups
: A collection of items or of other groups. One group in the sample diagram comprises the keywords FORM, PROC, QUERY, and TABLE. Another group in the sample comprises the two keywords YES and NO.

Items Individual elements inside the diagram. In the sample diagram, the items are keywords (the words shown in uppercase letters), a variable (the word *name*), a delimiter (the left parenthesis), and an operator (the = character). Items can also include fragment references and separators. These items needs to be in groups.

IBMIDDoc provides elements and attributes to mark up the syntax diagram elements. Here is the markup we used for the sample diagram:

```
<syntax><title>SAA CPI Database Reference</title>
<group>
<kwd>ERASE</kwd>
</group>
<group choiceseq="CHOICE">
<kwd>FORM</kwd>
<kwd>PROC</kwd>
<kwd>QUERY</kwd>
<kwd>TABLE</kwd>
</group>
<group>
<var>name</var>
</group>
<group optreq="OPT" choiceseq="composite"><delim>
(</delim><kwd>CONFIRM</kwd><oper>=</oper>
<group choiceseq="CHOICE">
<kwd>YES</kwd>
<kwd>NO</kwd>
</group>
</group>
</syntax>
```

IBMIDDoc also provides elements and attributes for elements not illustrated in the sample diagram.

Fragments
: A part of a syntax diagram, separated from the diagram to show greater detail. Like a syntax diagram, a fragment can contain items and groups. We do not mean to imply that the main syntax diagram is always complete. Often a main syntax diagram shows only a part of the syntax of the whole program. The word "fragment," as used here, means a part of your main diagram or of another fragment.

Syntax notes (SynNote)
: Notes often placed at the bottom of the diagram. Syntax notes are similar to footnotes placed in text.

RepSep
: Defines a repeat separator in a syntax diagram.

SynBlk
Organizes syntax definitions into subdivisions and keeps them together on a line.

SynPh Contains syntax elements, and is usually used to show a portion of a syntax definition.

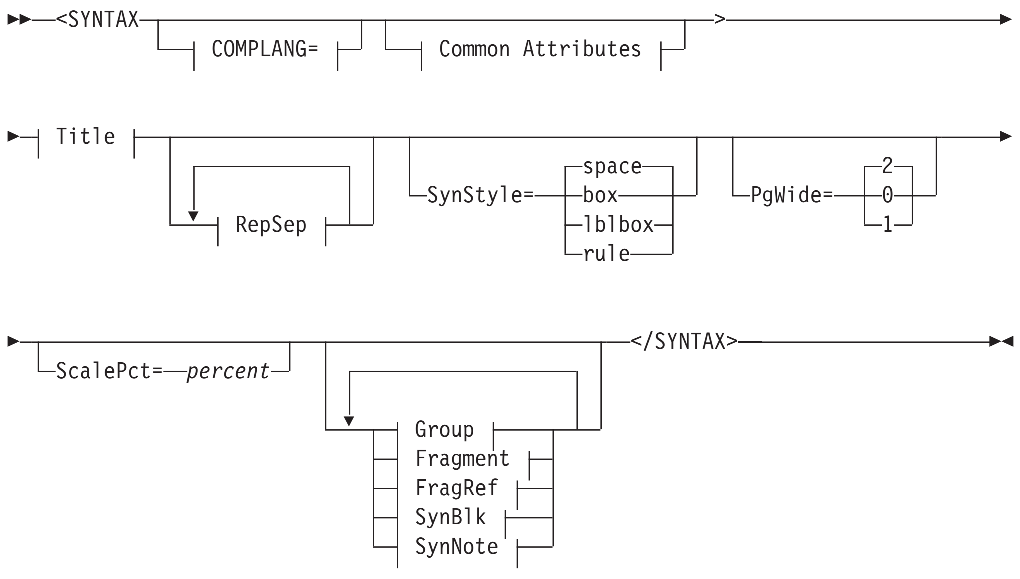
The Syntax element

The Syntax element contains the syntax diagram markup. The attributes of the Syntax element define the characteristics of the diagram.

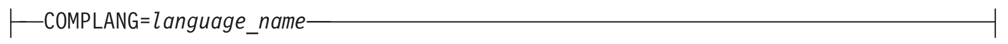
The text of the syntax diagram title can be contained in a Title element within the Syntax element. For example, we included a title, "SAA CPI Database Reference," within the Title element on our previous sample syntax diagram.

The following diagram shows the format for the Syntax element itself:

SYNTAX



COMPLANG=



For wide diagrams being output for BookManager BOOKs, specify a style override that uses the BookMaster DWIDTH attribute (the default value is 74):

```
<syntax style="bkm:(dwidth=100)">
```

The PgWide attribute controls the width of your diagram. 0 is page-wide, 1 is column-wide, and 2 (the default) is as wide as the current text line.

The ScalePct attribute allows you to scale a diagram up or down. For example, scalepct=150 makes the diagram 150% of the normal size.

You can have a box around your diagram, rules above and below it, or a labeled box around your diagram. Use the SynStyle attribute to add these style effects. The default is a space (SynStyle=Space).

SynStyle=LblBox
: Causes a box to be placed around the diagram. The top line of the box has text label that is taken from the diagram's Title tag.

SynStyle=Box
: Causes a box to be placed around the diagram.

SynStyle=Rule
: Causes a line to be placed above and below the diagram; to visually separate it from the surrounding text.

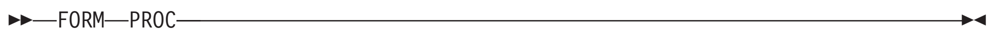
The Group element

The Group element defines the syntax group and lets you give the group a name in a Title element. Groups are needed to collect items like keywords, delimiters, variables into logical gatherings. Groupings indicate sequential items that need to be entered together or choices between mutialy exclusive items.

The Title element enables the Group to be automatically fragmented if it is too large to fit the current area. All items in a sequential group are kept on the same line. If you have several items that are too wide for one line, you'll have to split them into separate groups.

Each of the following examples shows a group with two keywords.

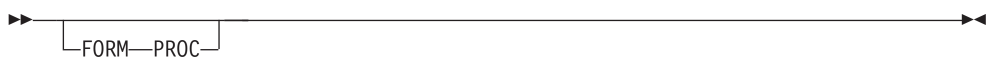
- In this example the group is required and sequential:

  Here's its markup:

  ```
  <syntax>
  <group>
  <kwd>FORM</kwd>
  <kwd>PROC</kwd>
  </group>
  </syntax>
  ```

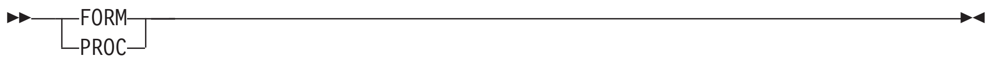
- In this example the group is sequential and optional (optreq attribute):

  Here's its markup:

  ```
  <syntax>
  <group optreq="opt">
  <kwd>FORM</kwd>
  <kwd>PROC</kwd>
  </group>
  </syntax>
  ```

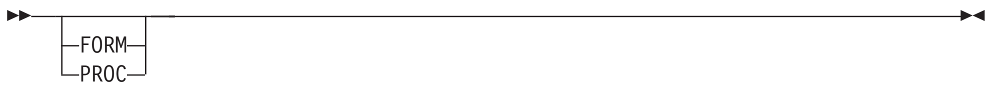
- In the next example, the group is a choice of the two keywords (choiceseq attribute); one is required:



Here’s its markup:

```
<syntax>
<group choiceseq="choice">
<kwd>FORM</kwd>
<kwd>PROC</kwd>
</group>
</syntax>
```

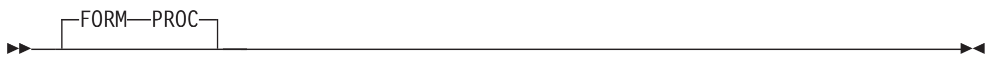
- In the next example, the group is a choice of the two keywords (choiceseq attribute) but they are optional (optreq attribute):



Here’s its markup:

```
<syntax>
<group optreq="opt" choiceseq="choice">
<kwd>FORM</kwd>
<kwd>PROC</kwd>
</group>
</syntax>
```

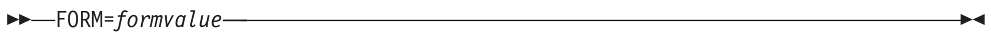
- In the next example, the group is a sequential default, the values are assigned even if you enter nothing:



Here’s its markup:

```
<syntax>
<group optreq="def" choiceseq="seq">
<kwd>FORM</kwd>
<kwd>PROC</kwd>
</group>
</syntax>
```

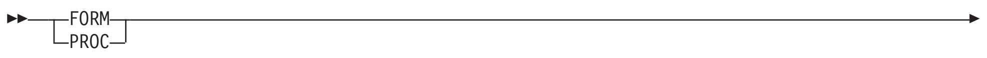
- Sometimes you need to show a diagram and indicate there is no interveining space between the items. The composite attribute means “sequential with no spaces”:



Here’s its markup:

```
<syntax>
<group choiceseq="composite"><kwd>FORM</kwd><delim>=</delim>
<var>formvalue</var></group>
</syntax>
```

- This shows an example of several sequential groups; this is to allow the diagram to break and flow properly:



```
►─printer──LPT1──┬─Portrait──┬──SOME LARGE KEYWORD TO GET THE DIAGRAM TO BREAK AND FLOW──────►◄
                 └─Landscape─┘
```

Here’s its markup:

```
<syntax>
<group choiceseq="choice">
<kwd>FORM</kwd>
<kwd>PROC</kwd>
</group>
<group>
<var>printer</var>
</group>
<group>
<kwd>LPT1</kwd>
</group>
<group choiceseq="choice">
<kwd>Portrait</kwd>
<kwd>Landscape</kwd>
</group>
<group>
<kwd>SOME LARGE KEYWORD TO GET THE DIAGRAM TO BREAK AND FLOW</kwd>
</group>
</syntax>
```

The KWD (keyword) element

The KWD element describes a keyword, which is a command name or any other literal information.

Examples:

- In this example, a group element contains two keywords, each contained in KWD elements:

```
►►──FORM──PROC──────────────────────────────────────────────────────────────►◄
```

Here’s its markup:

```
<syntax>
<group>
<kwd>FORM</kwd>
<kwd>PROC</kwd>
</group>
</syntax>
```

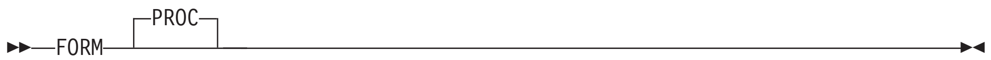
- In this example, the PROC keyword is optional:

```
►►──FORM──┬────────┬────────────────────────────────────────────────────────►◄
          └─PROC───┘
```

Here’s its markup:

```
<syntax>
<group>
<kwd>FORM</kwd>
<kwd optreq="opt">PROC</kwd>
</group>
</syntax>
```

- In this example, the PROC keyword is a default:



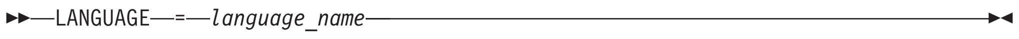
Here's its markup:

```
<syntax>
<group>
<kwd>FORM</kwd>
<kwd optreq="def">PROC</kwd>
</group>
</syntax>
```

The VAR (variable) element

The VAR element describes any variable information.

In this example, the VAR element contains the text `language_name`.



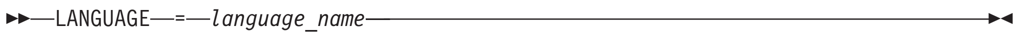
Here's its markup:

```
<syntax>
<group>
<kwd>LANGUAGE</kwd>
<oper>=</oper>
<var>language_name</var>
</group>
</syntax>
```

The OPER (operator) element

The OPER element describes an operator. Operators include add (+), subtract (-), multiply (*), divide (/), equal (=), and other mathematical operators. The operator can consist of more than one character.

In this example, the OPER element contains an equals (=) sign.

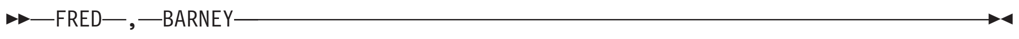


Here's its markup:

```
<syntax>
<group>
<kwd>LANGUAGE</kwd>
<oper>=</oper>
<var>language_name</var>
</group>
</syntax>
```

The SEP (separator) element

The SEP element describes a separator that is to separate keywords, variables, operators, or groups. The separator can be more than one character.



Here's its markup:

```
<syntax>
<group>
<kwd>FRED</kwd>
<sep>,</sep>
<kwd>BARNEY</kwd>
</group>
</syntax>
```

The Delim (delimiter) element

The Delim element specifies a delimiter that is to indicate the start or end of keywords, variables, operators, or groups. The delimiter can be one or more characters.

Examples:

- In this example, the delimiter is a plus (+) sign:

  ```
  ►►──FRED──+──WILMA──────────────────────────────────────────────────────►◄
  ```

  Here's its markup:

  ```
  <syntax>
  <group>
  <kwd>FRED</kwd>
  <delim>+</delim>
  <kwd>WILMA</kwd>
  </group>
  </syntax>
  ```

- You can use the STARTEND attribute to ensure that delimiters are specified in matched sets. If the syntax diagram requires a single delimiter, do not use STARTEND.

  ```
  ►►──ID(identifier)──────────────────────────────────────────────────────►◄
  ```

  Here's its markup:

  ```
  <syntax>
  <group choiceseq="composite"><kwd>ID</kwd><delim startend="start">
  (</delim><kwd>identifier</kwd><delim startend="end">
  )</delim></group>
  </syntax>
  ```

The RepSep (repeat separator) element

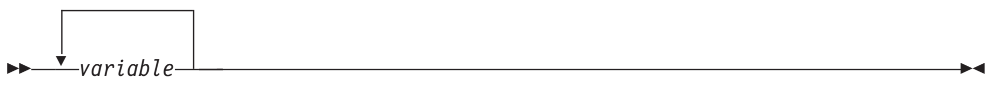
The RepSep element specifies whether the group of items or groups can repeat, and also the repeat separator character, if one is to be used. If the repeat separator character is specified, it separates the repeated group of items or groups in the syntax diagram.

The RepSep element must have an ID value. This ID is used when referencing the RepSep element from within the syntax markup. Use the REPID attribute on the repeating group; it references the ID on a RepSep element.

```
►►──<repsep─id=identifier──┬──────────────────┬──┬─────────────────────────┬──┤ common attributes ├────►◄
                           │          ┌─req─┐ │  │         ┌─constant─┐    │
                           └─optreq=──┴─opt─┴─┘  └─convar=─┴─variable─┴────┘
```

Examples:

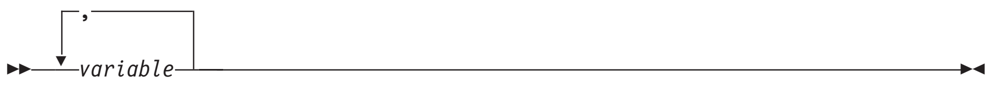
- The following example shows a group containing a variable you can repeat; there is no repeat separator character.



Here's its markup:

```
<syntax>
<repsep id="rsep0003a"></repsep>
<group repid="rsep0003a">
<var>variable</var>
</group>
</syntax>
```

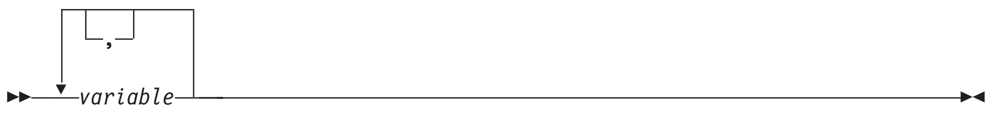
- The following example shows a group containing a variable and a repeat separator character. In this example, the repeat separator character is required:



Here's its markup:

```
<syntax>
<repsep id="rsep0003">,</repsep>
<group repid="rsep0003">
<var>variable</var>
</group>
</syntax>
```

- The following example shows a group containing a variable and a repeat separator character. Here the repeat separator character is optional.



Here's its markup:

```
<syntax>
<repsep optreq="OPT" id="rsep0004">,</repsep>
<group repid="rsep0004">
<var>variable</var>
</group>
</syntax>
```

The FRAGMENT and FRAGREF (fragment reference) element

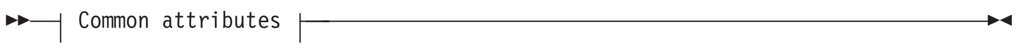
A syntax diagram can contain a section that has too many items or groups to fit in the diagram, or it can contain a section that is used more than once. You can present such a section as a separate fragment. You give the fragment a name that corresponds to the name of the section in the main diagram represented by the fragment.

The Fragment element specifies a fragment of your main syntax diagram or another fragment. The Fragment element is similar to the Syntax element. You can use Kwd, Var, Oper, Delim, Sep, FragRef, Group, and SynNote. These elements let you specify a diagram fragment in the same way that you specify a main diagram.

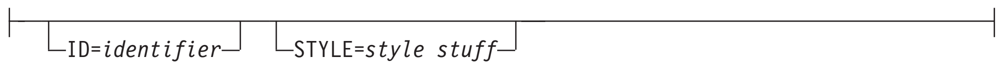
You can specify as many fragments as you want for a main diagram. The Fragment elements cannot be placed inside a Group element. Fragment is valid only within Syntax and SynBlk elements.

The FRAGREF element describes a reference to a syntax diagram fragment. The text of the FRAGREF element is placed in the syntax diagram and must match the name of the fragment reference that it refers to.

This shows a simple fragment that is done as a FragRef and a Fragment:



Common attributes:



Here’s its markup:

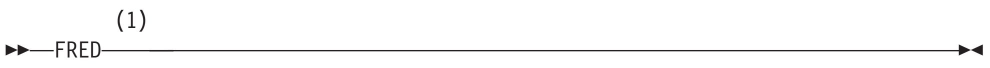
```
<syntax>
<fragref><title>Common attributes</title></fragref>
<fragment><title>Common attributes</title>
<group optreq="opt" choiceseq="composite"><kwd>ID
</kwd><oper>=</oper><var>identifier</var></group>
<group optreq="opt" choiceseq="composite"><kwd>STYLE
</kwd><oper>=</oper><var>style stuff</var></group>
</fragment>
</syntax>
```

Syntax Notes

IBMIDDoc provides the SynNote element for placing notes in your syntax diagrams. Syntax notes are similar to footnotes in regular text. At processing time, a number or other callout is placed next to an item, group, or fragment in the diagram, indicating that a note is associated with that part of the diagram, and the note appears at the bottom of the diagram, after any fragments.

Examples:

- This shows a simple note:



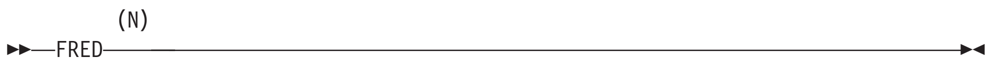
Notes:

1 This is a rather common name.

Here’s its markup:

```
<syntax>
<group>
<kwd>FRED</kwd>
<synnote>This is a rather common name.</synnote>
</group>
</syntax>
```

- You can use the CALLOUT attribute in hardcopy to have a specific character displayed for the note item. This diagram is brought to you by the letter ″N″:

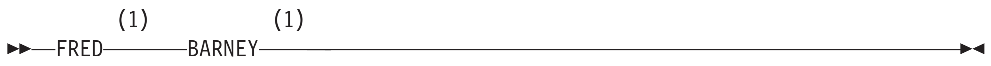


Notes:

N This is a rather common name.
Here’s its markup:

```
<syntax>
<group>
<kwd>FRED</kwd>
<synnote callout="N">This is a rather common name.
</synnote>
</group>
</syntax>
```

- You can also specify a note once, then refer to it more than once:



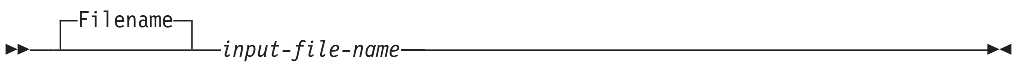
Notes:

1 This is a rather common name.
Here’s its markup:

```
<syntax>
<synnote id="comname">This is a rather common name.
</synnote>
<group>
<kwd>FRED</kwd>
<synnote refid="comname">
</group>
<group>
<kwd>BARNEY</kwd>
<synnote refid="comname">
</group>
</syntax>
```

Syntax Phrases

Syntax phrases allow you to use a portion of a syntax statement; such as a term in a parameter list. For example:



Filename
Sample description for this syntax item.

This is the markup:

```
<syntax>
<group>
<kwd optreq="def">Filename</kwd>
<var>input-file-name</var>
</group>
</syntax><parml>
```

```
<parm><term><synph><kwd optreq="def">Filename</kwd></synph></term>
<defn>Sample description for this syntax item.</defn>
</parm>
</parml>
```

Examples of Syntax Definitions and Markup

The examples in the sections that follow represent typical syntax definitions.

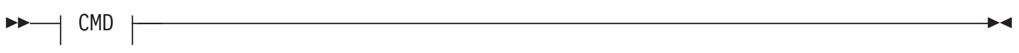
Example 1: A simple syntax definition

This example illustrates one of the simplest styles of syntax definition, with only one optional parameter value.

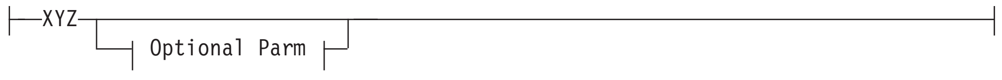
```
<SYNTAX>
 <TITLE>XYZ Command</TITLE>
  <GROUP>
   <TITLE>CMD</TITLE>
    <KWD>XYZ</KWD>
     <GROUP OPTREQ="OPT">
      <TITLE>OPTION 1</TITLE>
       <SEP>&ssbl;</SEP>
       <KWD>PARM</KWD>
       <OPER>&equals;</OPER>
       <VAR>value</VAR>
     </GROUP>
  </GROUP>
</SYNTAX>
```

This SGML input will produce the following output.

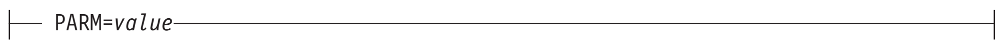
XYZ Command



CMD



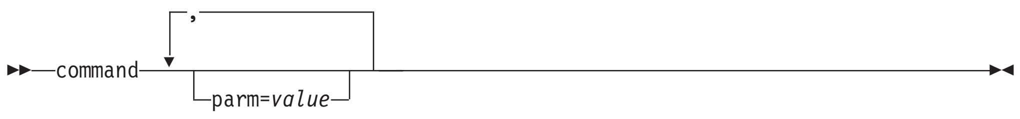
Optional Parm



Example 2: A simple syntax definition that repeats

This example illustrates a syntax definition for a command with a parameter that can be repeated.

Syntax Diagram With Repetition



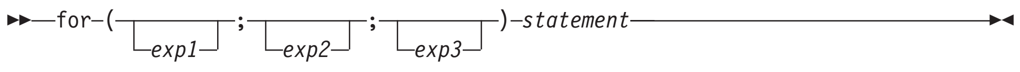
Here's its markup:

```
<syntax><title>Syntax Diagram With Repetition</title>
<repsep id="REP1">,</repsep>
<group>
<kwd>command</kwd>
</group>
<group repid="REP1" optreq="OPT" choiceseq="composite">
<kwd>parm</kwd><oper>=</oper><var>value</var></group>
</syntax>
```

Example 3: A more complex syntax definition

The following syntax definition contains a single group comprising three variable expressions, two separators, and two delimiters. Each variable expression is optional. The definition also includes a required keyword and a required variable statement.

SAA CPI C Reference



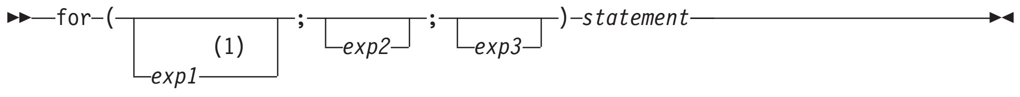
Here's its markup:

```
<syntax><title>SAA CPI C Reference</title>
<group choiceseq="composite"><kwd>for</kwd>
<group choiceseq="composite"><delim optreq="req" startend="START">
(</delim><var optreq="OPT">exp1</var><sep optreq="req">
&semi;</sep><var optreq="OPT">exp2</var><sep optreq="req">
&semi;</sep><var optreq="OPT">exp3</var><delim optreq="req"
startend="END">)</delim></group>
<var>statement</var></group>
</syntax>
```

Example 4: A variation on Example 3

This is the same diagram as in "Example 3: A more complex syntax definition", with a syntax note added.

SAA CPI C Reference



Notes:

1 This indicates the beginning condition.

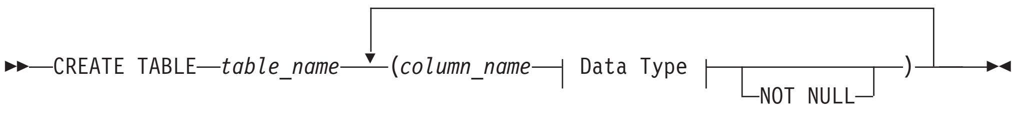
Here's its markup:

```
<syntax><title>SAA CPI C Reference</title>
<group choiceseq="composite"><kwd>for</kwd>
<group choiceseq="composite"><delim optreq="req" startend="START">
(</delim><var optreq="OPT">exp1</var><sep optreq="req">
&semi;</sep><var optreq="OPT">exp2</var><sep optreq="req">
&semi;</sep><var optreq="OPT">exp3</var><delim optreq="req"
startend="END">)</delim></group>
<var>statement</var></group>
</syntax>
```

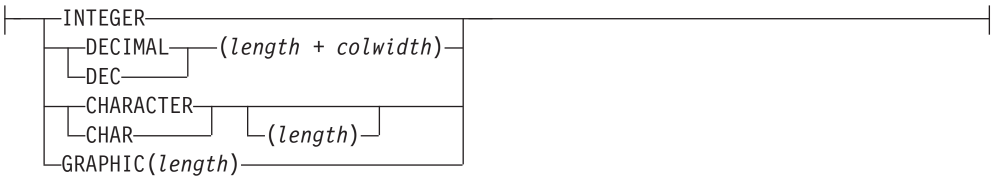
Example 5: A syntax definition showing a fragment and significant blanks

The following syntax definition includes a fragment called "Data Type." The fragment is placed below the main syntax definition. This example also shows the use of the syntax significant blank symbol (&ssbl.); use this to ensure a blank is left in the diagram where the user should code a space.

Database Reference



Data Type:



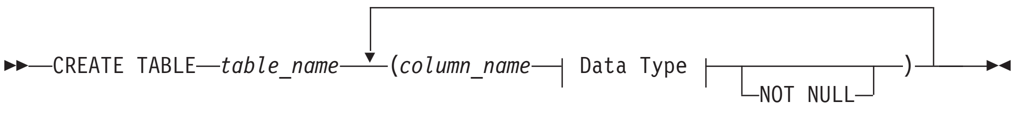
Here's its markup:

```
<syntax><title>Database Reference</title>
<repsep id="rsep0006"></repsep>
<group>
<kwd>CREATE TABLE</kwd>
</group>
<group>
<var>table_name</var>
</group>
<group repid="rsep0006">
<group choiceseq="composite"><delim startend="START">
(</delim><var>column_name</var></group>
<fragref><title>Data Type</title></fragref>
<kwd optreq="OPT">NOT NULL</kwd>
<delim optreq="req" startend="END">)</delim>
</group>
<fragment><title>Data Type</title>
<group choiceseq="CHOICE">
<kwd>INTEGER</kwd>
<group>
<group choiceseq="CHOICE">
<kwd>DECIMAL</kwd>
<kwd>DEC</kwd>
</group>
<group choiceseq="composite"><delim startend="START">
(</delim><var>length</var><sep>&ssbl;+&ssbl;</sep>
<var>colwidth</var><delim startend="END">)</delim>
</group>
</group>
<group>
<group choiceseq="CHOICE">
<kwd>CHARACTER</kwd>
<kwd>CHAR</kwd>
</group>
<group optreq="OPT" choiceseq="composite"><delim startend="START">
(</delim><var>length</var><delim startend="END">)
</delim></group>
</group>
<group choiceseq="composite"><kwd>GRAPHIC</kwd><delim
startend="START">(</delim><var>length</var><delim
startend="END">)</delim></group>
</group>
</fragment>
</syntax>
```

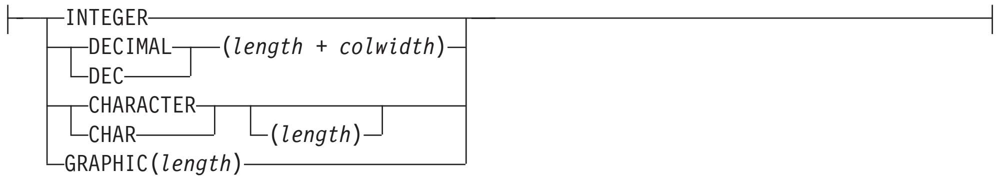
Example 6: A syntax definition with automatic fragmenting

The following example is identical to "Example 5: A syntax definition showing a fragment and significant blanks" on page 160 except that the fragment is marked up as a group with a title. Because the group is very wide, it automatically fragments.

Database Reference



Data Type



Here's its markup:

```
<syntax><title>Database Reference</title>
<repsep id="rsep0006"></repsep>
<group>
<kwd>CREATE TABLE</kwd>
</group>
<group>
<var>table_name</var>
</group>
<group repid="rsep0006">
<group choiceseq="composite"><delim startend="START">
(</delim><var>column_name</var></group>
<fragref><title>Data Type</title></fragref>
<kwd optreq="OPT">NOT NULL</kwd>
<delim optreq="req" startend="END">)</delim>
</group>
<fragment><title>Data Type</title>
```

```
<group choiceseq="CHOICE">
<kwd>INTEGER</kwd>
<group>
<group choiceseq="CHOICE">
<kwd>DECIMAL</kwd>
<kwd>DEC</kwd>
</group>
<group choiceseq="composite"><delim startend="START">
(</delim><var>length</var><sep>&ssbl;+&ssbl;</sep>
<var>colwidth</var><delim startend="END">)</delim>
</group>
</group>
<group>
<group choiceseq="CHOICE">
<kwd>CHARACTER</kwd>
<kwd>CHAR</kwd>
</group>
<group optreq="OPT" choiceseq="composite"><delim startend="START">
(</delim><var>length</var><delim startend="END">)
</delim></group>
</group>
<group choiceseq="composite"><kwd>GRAPHIC</kwd><delim
startend="START">(</delim><var>length</var><delim
startend="END">)</delim></group>
</group>
</fragment>
</syntax><syntax><title>Database Reference</title>
<repsep id="rsep00061"></repsep>
<group>
<kwd>CREATE TABLE</kwd>
</group>
<group>
<var>table_name</var>
</group>
<group repid="rsep00061">
<group choiceseq="composite"><delim startend="START">
(</delim><var>column_name</var></group>
<group choiceseq="CHOICE"><title>Data Type</title>
<kwd>INTEGER</kwd>
<group>
<group choiceseq="CHOICE">
<kwd>DECIMAL</kwd>
<kwd>DEC</kwd>
</group>
<group choiceseq="composite"><delim startend="START">
(</delim><var>length</var><sep>&ssbl;+&ssbl;</sep>
<var>colwidth</var><delim startend="END">)</delim>
</group>
</group>
<group>
<group choiceseq="CHOICE">
<kwd>CHARACTER</kwd>
<kwd>CHAR</kwd>
</group>
<group optreq="OPT" choiceseq="composite"><delim startend="START">
(</delim><var>length</var><delim startend="END">)
</delim></group>
</group>
<group choiceseq="composite"><kwd>GRAPHIC</kwd><delim
startend="START">(</delim><var>length</var><delim
startend="END">)</delim></group>
</group>
<group>
<kwd optreq="OPT">NOT NULL</kwd>
<delim optreq="req" startend="END">)</delim>
</group>
</group>
</syntax>
```

Chapter 16. Developing Programming Language Reference Materials

We are often called upon to produce reference information for the various elements of programming languages, either as the major portion of a language reference manual, or as part of a combined language guide and reference. In this context we use the term "programming language" very broadly, including the higher-level languages (such as Java), command and control languages (such as JCL, TSO, and CMS commands), macro languages, such as Access Method Services macros, and markup languages (such as our very own IBMIDDoc).

These reference materials typically have a couple of things in common:
- The same set of subtopics (format, parameters, usage, and so on) is repeated for each language element (statement, macro, and so on). While each subtopic may have its own heading, you don't want these headings to appear in your table of contents.
- Consistency of presentation and retrievability are critical, as readers want to find the information as quickly as possible and not have to re-interpret the presentation each time.

Another good reason to use IBMIDDoc elements for language element reference materials is that once you've determined how you're going to use the elements for your particular language reference material, you'll find they are a help in developing consistent, well-structured materials that are easier to maintain.

The Structure of a Language Element Reference Section

Use LERS to contain reference information for computer languages and command information. LERS contains one or more Language Elements (LEs) that contain the description of a computer language element such as commands, and description items, such as format, purpose, and examples. There are several elements used to complete the LERS section.

A typical LERS section looks like:

```
language element reference section <LERS>
  language element <LE>
  language element name <LEN>
  language element description <LEDesc>
  language element description item <LEDI>

⋮
  language element <LE>
  language element name <LEN>
  language element description <LEDesc>
  language element description item <LEDI>

⋮
end language element reference section </LERS>
```

The language element reference section (LERS) contains many language elements (described with the LE, LEN, and LEDESC tags), and for each of the language elements you can have lots of description items (LEDI tags). By description items, we mean such things as format (sometimes called "syntax"), purpose, examples — those categories of information we typically provide when describing a language element.

Describing Your Reference Section

Your first task in creating a language element reference section (after you figure how you're going to present the material, of course) is to describe it, which you do with attributes on either the LERS (language element reference section) element or the LERSDEF (LERS definition) element.

While a book typically might have only a few, or perhaps just one language element reference section, these sections can be enormously long. It is impractical and inefficient to handle these long sections in a single source file; you will want to break up the material into multiple files, each with its own LERS element, so that each file can be processed independently. This is why we have the LERSDEF element. It allows you to specify all of the LERS attributes — you put the LERSDEF in your Prolog, and it is referenced in your document.

What can you describe on the LERS or LERSDEF element about your reference section?

- **How to get at the description**

  The LERSDEF element has a DEFNAME attribute so you can refer to it using the DEF attribute on the LERS element. However, if you are setting your document's default (all your LERS are the same), you can use LERSDEF in the Prolog without the DEFNAME attribute.
- **What text you want generated for the description items**

  Each language element description item element (LEDI) has an attribute that defines the category of that description item. IBMIDDoc will generate a subheading for each description item, as determined by the attribute.

  For each category, IBMIDDoc has a corresponding attribute on the LERS and LERSDEF elements that allows you to specify the subheading text you want in place of the text generated by default. You can specify any text you want, or you can specify that *no heading be generated at all*.
- **Whether each language element starts a new page**

  Normally, each language element begins on a new page. To control the separation between language elements, you use the SEP attribute on the LERS or LERSDEF tag. You can pick from the following:

  SEP= **PAGE** **|** **NORMAL** **|** **LHPAGE** **|** **RHPAGE**
  allows you to specify how you want the language elements separated, where:

  PAGE Starts the language element on the next page.

  NORMAL
  Specifies normal heading separation — usually white space.

  LHPAGE
  Starts the language element on the next left-hand page (even page).

  RHPAGE
  Starts the language element on the next right-hand page (odd page).
- **Whether the language elements are to be used as the retrieval subject for a page**

It may be easier for a reader to look up a language element in a long list if the language elements are used as retrieval subjects. In styles where the subject is placed in the running heading, such as the default style, this results in dictionary-like running headings. You can pick from the following:

RETKEY=None | First

Use the RetKey attribute to enable or disable automatic running headings for this tag.

NONE

indicates that nothing is to be used.

FIRST

indicates that the first non-blank item on the page is to be used.

The values NODUP and FIRSTLAST are not supported at this time.

If you code any explicit RetKey elements, they are honored and will appear. If you nest elements that can generate a running heading (for example, a MsgList inside Lers), only the outer active generated heading is used. That is, if you specified automated RetKey generation for LERS and MSGLIST, a Msgno inside Lers will not be used in the RetKey area. But if you had an explicit RetKey inside the Msg, then the RetKey is honored as an explicit override.

Table 17 shows the attribute name for each of the language element description item categories, the text that will be generated by default in styles that use IBMIDDoc's initial setting, and what the category means. Of course, you will only use the categories that are appropriate for your material.

Table 17. Categories of language element description items

Attribute name	Generated heading	Description
AUTH	Authorization	the authorization level necessary to use this language element
COMMENTS	Comments	just about anything you consider comments
CONTEXT	Context	the context in which this language element is valid
DEFAULTS	Defaults	the defaults
ERRCOND	Error Conditions	error conditions that can arise from misuse
EXAMPLES	Examples	examples of input and output
FLAGS	Flags	the flags that could be set by the language element
FORMAT	Format	the general format (or syntax)
INTREP	Internal Representation	the internal representation (for example, binary) of the language element (sometimes called "encoding")
MESSAGES	Messages	messages that can be generated as a result of use of this element
OTHER		a build-your-own category
PARMS	Parameters	the parameters of the language element
PROCESS	Processing	the processing that will be done for the language element (that is, the logic)

Table 17. Categories of language element description items (continued)

Attribute name	Generated heading	Description
PURPOSE	Purpose	the purpose of the language element
RESTRICT	Restrictions	restrictions on use of the language element
RESULTS	Return Codes	explanations of the return codes possible with the language element
SYSENV	System Environment	the system environment in which the language element is valid
USAGE	Usage	how the language element is used
VERSION	Version	the version of the program in which the language element is valid

The categories have been selected based on a review of what has been used in the past. The OTHER category exists to allow you to create a category that has not been anticipated in the list above; before you decide to use OTHER to create a new category, review the ones available carefully to make sure that your category doesn't already exist.

Be assured that IBMIDDoc is not demanding that you structure your reference information to match these categories. You might very well want to deal with the defaults as part of the discussion of the parameters, rather than as a separate "Defaults" category; similarly, you might want to deal with "Context" in the discussion of usage. These decisions depend on the nature of the language you are describing and the approach you take to its presentation.

So you might, for example, decide the following:

- Your reference section should have the categories PURPOSE, FORMAT, PARMS, USAGE, and EXAMPLES
- You do want the FORMAT category headed "Syntax"
- You do want the PARMS category headed "Attributes and Contained Elements"

Your LERSDEF might look like this:

```
<LERSDEF
 FORMAT="Syntax"
 PARMS="Attributes and Contained Elements"
 DEFNAME="UGREFLERS">
 <DESC>Contains the LEDI IBMIDDoc User's Guide
  and Reference LERS name specifications.</DESC>
</LERSDEF>
```

You didn't have to say anything about PURPOSE or EXAMPLES because you want the default headings generated.

Describing the language element

You start each item in a LERS section with an LE (language element) tag. It is like a division tag. The LEN (language element name) tag contains the title of the LE. You can format where the line breaks occur in the title if you would like. For example, the following will format the title over two lines:

```
<len>COPYFILE
Copy a file
</len>
```

In the table of contents, the line split does not occur (except in Frame2000 currently).

Following the language element name, you have a series of language element description items, marked up with the LEDI element. Each LEDI element CLASS attribute needs to have an attribute that describes the category of the item:

```
AUTH       EXAMPLES   OTHER      RESULTS
COMMENTS   FLAGS      PARMS      RETCODES
CONTEXT    FORMAT     PROCESS    SYSENV
DEFAULTS   INTREP     PURPOSE    USAGE
ERRCOND    MESSAGES   RESTRICT   VERSION
```

These are the same description item category attributes that occur on the LERS and LERSDEF element (see Table 17 on page 167).

Example of a Simple Language Element Reference Section

Time for an example. In this example, we use these categories:

PURPOSE
Without a heading. Because it follows the LEN immediately, it doesn't need its own heading.

FORMAT
Uses the special heading text "Syntax".

PARMS
Without a heading. It appears to the reader as simply part of the syntax discussion.

USAGE
Uses the default heading text.

RESTRICT
Uses the special heading text "Do's and Don'ts".

EXAMPLES
Uses the default heading text.

MESSAGES
Uses the default heading text. We only have one command that yields messages, so this is used only once.

Our language in the following example is a command language for a culinary robot. Here's its coding:

```
<lers format="Syntax" parms="" process="" restrict="Do's and Don'ts">
<le>
<len>DISHDEF
defining a dish</len>
<ledi class="purpose">
<p>The DISHDEF command defines a dish &mdash; the
ingredients that it contains and the processing steps
to prepare it.</p>
</ledi>
<ledi class="format">
<syntax>
<repsep id="cul"></repsep>
<group>
<kwd>DISHDEF</kwd>
</group>
<group style="bkm:(composite)">
<kwd>NAME</kwd>
<delim>=</delim>
```

```
<var>name-of-dish</var>
</group>
<group repid="cul" style="bkm:(composite)">
<kwd>INGREDIENT</kwd>
<delim>=</delim>
<var>ingredient-name</var>
<delim>/</delim>
<var>quantity</var>
</group>
<group repid="cul" style="bkm:(composite)">
<kwd>STEP</kwd>
<delim>=</delim>
<var>process-name</var>
<delim> (</delim>
<var>ingredient-list</var>
<delim>)</delim>
<group choiceseq="choice">
<group>
<kwd>UNTIL</kwd>
<var>condition-name</var>
</group>
<group>
<kwd>FOR</kwd>
<var>time</var>
</group>
</group>
</group>
</syntax>
</ledi>
<ledi class="parms">
<parml>
<parm><term><synph><kwd>NAME</kwd><delim>=</delim><var>
name-of-dish</var></synph></term>
<defn>identifies the dish name.</defn>
</parm>
<parm><term><synph><kwd>INGREDIENT</kwd><delim>=</delim><var>
ingredient-name</var><delim>/</delim><var>quantity
</var></synph></term>
<defn>identifies an ingredient and the quantity per
serving. The ingredient must be expressed in international
culinary ingredient units (ICIUs). The quantity must
be expressed in international culinary quantity units
(ICQUs). This parameter is repeated as often as necessary
to define each of the ingredients in the dish.</defn>
</parm>
<parm><term><synph><kwd>STEP</kwd><delim>=</delim><var>
process-name</var><delim>(</delim><var>ingredient-list
</var><delim>)</delim></synph></term>
<defn>identifies a preparation step and the ingredients
to use. Process names must be expressed in international
culinary step units (ICSUs). This clause is repeated
as often as necessary to define each of the preparation
steps for the dish.</defn>
</parm>
<parm><term><synph><kwd>UNTIL</kwd><var>condition-name
</var></synph></term><term><synph><kwd>FOR</kwd><var>
time</var></synph></term>
<defn>identifies a condition under which the step
is to conclude or an amount of time for processing.
Condition names must be expressed in international
culinary condition units (ICCUs). </defn>
</parm>
</parml>
</ledi>
<ledi class="usage">
<p>Use the DISHDEF command to specify to the culinary
robot how to prepare a dish. </p>
```

```
</ledi>
<ledi class="restrict">
<p>Make sure that the ingredients list is in the order
in which the ingredients are to be used. For example,
for a SAUTE step, make sure that BUTTER is specified
before VEAL or the robot will put the veal in the
pan before the butter.</p>
</ledi>
<ledi class="examples">
<xmp>dishdef name=vealalfred
ingredient=butter/1tsp
ingredient=vealscallop/6oz ingredient=salt/pinch
ingredient=tarragon/1tsp ingredient=sourcream/halfcup
step=saute (butter vealscallop) until golden brown
step=add (salt tarragon) for 1 min
step=deglaze (sourcream) for 6 min</xmp>
</ledi>
</le>
<le>
<len>EVALUATE
evaluate nutrition, cost, or preparation time</len>
<ledi class="purpose">
<p>Once you have a menu defined, use the EVALUATE
command to determine its nutritional characteristics
and the preparation time. If you have access to the
Daily Market Cost data base, you can also evaluate
the cost of a shopping list containing one or more
menus.</p>
</ledi>
<ledi class="format">
<syntax>
<group>
<kwd>EVALUATE</kwd>
</group>
<group choiceseq="choice">
<group>
<kwd>NUTRITION</kwd>
<group style="bkm:(composite)">
<kwd>MENU</kwd>
<delim>=</delim>
<var>menu-name</var>
</group>
</group>
<group>
<kwd>COST</kwd>
<group style="bkm:(composite)">
<kwd>SHOPLIST</kwd>
<delim>=</delim>
<var>shopping-list-name</var>
</group>
</group>
<group>
<kwd>PREPTIME</kwd>
<group style="bkm:(composite)">
<kwd>MENU</kwd>
<delim>=</delim>
<var>menu-name</var>
</group>
<group style="bkm:(composite)">
<kwd>SERVING</kwd>
<delim>=</delim>
<var>number</var>
</group>
</group>
</group>
</syntax>
</ledi>
```

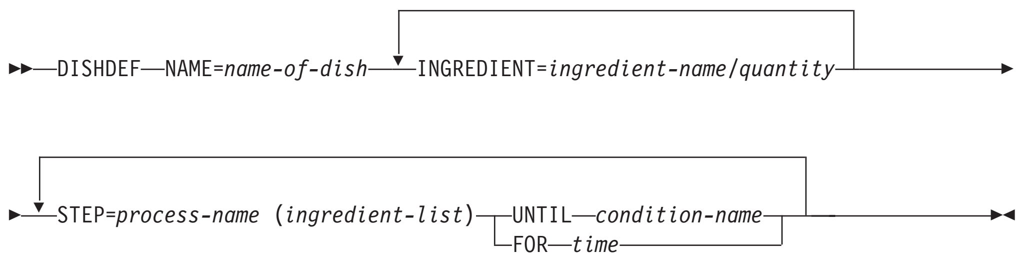
```
<ledi class="parms">
<parml>
<parm><term><synph><kwd>MENU</kwd><delim>=</delim><var>
menu-name</var></synph></term>
<defn>requests a nutritional evaluation of menu name.
</defn>
</parm>
<parm><term><synph><kwd>COST</kwd> <kwd>SHOPLIST</kwd><delim>
=</delim><var>shopping-list-name</var></synph></term>
<defn>as determined by your marketing profile and
the Daily Market Cost data base, this command generates
a cost for the named shopping list for each of the
markets in the profile, including the cost of the
gasoline for driving to those markets designated in
your profile as not providing delivery service.</defn>
</parm>
<parm><term><synph><kwd>PREPTIME</kwd> <kwd>MENU</kwd><delim>
=</delim><var>menu-name</var> <kwd>SERVING</kwd><delim>
=</delim><var>number</var></synph></term>
<defn>requests an evaluation of the preparation time
for the designated menu serving the designated number
of people.</defn>
</parm>
</parml>
</ledi>
<ledi class="usage">
<p>Use the EVALUATE command as required to maximize
the nutrition and minimize the cost of meals. Knowing
the preparation time is critical in requesting that
menus be prepared.</p>
</ledi>
<ledi class="examples">
<xmp>//evaluate nutrition menu=companydinner
//evaluate cost shoplist=monday
//evaluate preptime menu=companydinner serving=8</xmp>
</ledi>
</le>
</lers>
```

DISHDEF
defining a dish

Purpose

The DISHDEF command defines a dish — the ingredients that it contains and the processing steps to prepare it.

Syntax


DISHDEF
NAME=name-of-dish
INGREDIENT=ingredient-name/quantity
STEP=process-name (ingredient-list)
UNTIL
condition-name
FOR
time

NAME=*name-of-dish*
identifies the dish name.

INGREDIENT=*ingredient-name/quantity*

identifies an ingredient and the quantity per serving. The ingredient must be expressed in international culinary ingredient units (ICIUs). The quantity must be expressed in international culinary quantity units (ICQUs). This parameter is repeated as often as necessary to define each of the ingredients in the dish.

STEP=*process-name*(*ingredient-list*)

identifies a preparation step and the ingredients to use. Process names must be expressed in international culinary step units (ICSUs). This clause is repeated as often as necessary to define each of the preparation steps for the dish.

UNTIL*condition-name*
FOR*time*

identifies a condition under which the step is to conclude or an amount of time for processing. Condition names must be expressed in international culinary condition units (ICCUs).

Usage

Use the DISHDEF command to specify to the culinary robot how to prepare a dish.

Do’s and Don’ts

Make sure that the ingredients list is in the order in which the ingredients are to be used. For example, for a SAUTE step, make sure that BUTTER is specified before VEAL or the robot will put the veal in the pan before the butter.

Examples

```
dishdef name=vealalfred
ingredient=butter/1tsp
ingredient=vealscallop/6oz ingredient=salt/pinch
ingredient=tarragon/1tsp ingredient=sourcream/halfcup
step=saute (butter vealscallop) until golden brown
step=add (salt tarragon) for 1 min
step=deglaze (sourcream) for 6 min
```

EVALUATE
evaluate nutrition, cost, or preparation time

Purpose

Once you have a menu defined, use the EVALUATE command to determine its nutritional characteristics and the preparation time. If you have access to the Daily Market Cost data base, you can also evaluate the cost of a shopping list containing one or more menus.

Syntax

```
►►──EVALUATE──┬─NUTRITION──MENU=menu-name──────────────────┬──────────────────────────►◄
              ├─COST──SHOPLIST=shopping-list-name───────────┤
              └─PREPTIME──MENU=menu-name──SERVING=number────┘
```

MENU=*menu-name*

requests a nutritional evaluation of menu name.

COST SHOPLIST=*shopping-list-name*

as determined by your marketing profile and the Daily Market Cost data base, this command generates a cost for the named shopping list for each of the markets in the profile, including the cost of the gasoline for driving to those markets designated in your profile as not providing delivery service.

PREPTIME MENU=*menu-name* **SERVING**=*number*
requests an evaluation of the preparation time for the designated menu serving the designated number of people.

Usage

Use the EVALUATE command as required to maximize the nutrition and minimize the cost of meals. Knowing the preparation time is critical in requesting that menus be prepared.

Examples

```
//evaluate nutrition menu=companydinner
//evaluate cost shoplist=monday
//evaluate preptime menu=companydinner serving=8
```

Chapter 17. Defining Modular Information

Use modular information describe information that is often repetitive in structure and content, and requires precise markup. To do this, you can define many different modular information classes for each type of information you need to describe. You can use modular information to create reference information for most anything. This chapter introduces ways to create modular information using IBMIDDoc.

You can create modular information classes that describe programming data structures, commands and syntax, or command definitions, for example. This information is often displayed in a tabular format. You can define this format using the modular information elements described in this section.

Tables are a presentation-oriented structure. Modular information attempts to capture the relationships that are often expressed in tables, using classes of information, in a more meaningful way. Many tables have aspects with specific meanings. There are times when, for presentation-specific purposes, a table must be altered or resized—to fit on a page, for example. These changes can cause confusion about the relationships the table is intended to illustrate.

Modular information elements capture these meanings using the class mechanism that is part of modular item specifications. IBMIDDoc's modular information elements are a way to express many of the structures that are currently expressed as tables. IBMIDDoc modular information can be expressed in many ways, without obscuring the meaning of the relationships expressed in the information's classes.

You can use several elements in combination to define modular information classes, descriptions, and properties. These elements include:

- ModInfo
- ModInfoDef
- Mod
- ModItem
- ModItemDef

The following example illustrates how to use these elements to define modular information. See the reference section entries for the elements used in the example that follows for more information about these elements.

```
   ⋮
<PROLOG>
   ⋮
<PROPDEFS>
   ⋮
<MODINFODEF CLASSNAME="payobj">
 <DESC>This class of modular information should be used to
 describe payroll objects on a data entry screen.
 </DESC>
 <MODITEMDEF CLASSNAME="exempt">Exemptions
  <DESC>Contains the number of exemptions the employee claims.
 <MODITEMDEF CLASSNAME="rate">Hourly Rate
  <DESC>Contains the amount, in dollars and cents, the employee is
        paid per hour.
</MODINFODEF>
```

```
   ⋮
</PROPDEFS>
   ⋮
</PROLOG>
   ⋮
```

After defining the payobject, exemptions, and rate classes, they can be used as shown in the following example.

```
<MODINFO CLASS="payobj">
<MOD ID="empinf">
<MODNAME>Employee Pay Information</MODNAME>
<MODITEM CLASS="exempt">
<P>Enter the number of exemptions that the employee claims.</P></MODITEM>
<MODITEM CLASS="rate">
<P>Enter the employee's hourly rate of pay.</P></MODITEM>
</MOD>
   ⋮
</MODINFO>
```

Examples of Using Modular Information

Defining modular information allows the information to be displayed in a variety of ways, including a table presentation style. The example that follows contains modular information elements and a formatted example of one way to express the meaning of the modular information in a table presentation.

Here are the modular information definitions:

```
<propdefs>
<modinfodef classname="CUST"><desc>Customer information
</desc>
<moditemdef classname="NAME"><title>Name</title><desc>
This is the first and last name of the customer</desc>
</moditemdef>
<moditemdef classname="CUSTID"><title>ID</title><desc>
This is their customer id number</desc></moditemdef>
<moditemdef classname="INC"><title>Income</title>
<desc>This is their annual estimated income</desc>
</moditemdef>
<moditemdef classname="LPD"><title>Last purchase date
</title><desc>This is their last purchase date</desc>
</moditemdef>
<moditemdef classname="NOTES"><title>Notes</title>
<desc>Personal notes</desc></moditemdef>
</modinfodef>
</propdefs>
```

Here is the mudular information section:

```
<modinfo class="cust" style="table">
<mod class="CUST">
<modname class="NAME">Fred Smith</modname>
<moditem class="CUSTID"><p><num base="10">1000</num></p></moditem>
<moditem class="INC"><num base="10">40000</num></moditem>
<moditem class="LPD"><p><date>12/25/93</date></p></moditem>
<moditem class="NOTES"><p>Big spender</p></moditem>
</mod>
<mod class="CUST">
<modname class="NAME">Suzanne Stanley</modname>
<moditem class="CUSTID"><p><num base="10">1001</num></p></moditem>
<moditem class="INC"><p><num base="10">50000</num></p></moditem>
<moditem class="LPD"><p><date>11/22/92</date></p></moditem>
<moditem class="NOTES"><p>Likes game software</p></moditem>
</mod>
```

```
<mod class="CUST">
<modname class="NAME">Jeff George</modname>
<moditem class="CUSTID"><p><num base="10">1002</num></p></moditem>
<moditem class="INC"><p><num base="10">60000</num></p></moditem>
<moditem class="LPD"><p><date>12/02/93</date></p></moditem>
<moditem class="NOTES"><p>Likes DVD movies</p></moditem>
</mod>
<mod class="CUST">
<modname class="NAME">Mike Gidento</modname>
<moditem class="CUSTID"><p><num base="10">1003</num></p></moditem>
<moditem class="INC"><p><num base="10">35000</num></p></moditem>
<moditem class="LPD"><p><date>12/12/92</date></p></moditem>
<moditem class="NOTES"><p>Likes 8-track tapes</p></moditem>
</mod>
</modinfo>
```

The result looks like the following:

Name	**ID**	**Income**	**Last purchase date**	**Notes**
Fred Smith	1000	40000	12/25/93	Big spender
Suzanne Stanley	1001	50000	11/22/92	Likes game software
Jeff George	1002	60000	12/02/93	Likes DVD movies
Mike Gidento	1003	35000	12/12/92	Likes 8-track tapes

Chapter 18. File, text, and character entities and reusing information

Where multiple output documents contain common information, or a single document repeats the same information, the only way to ensure that the common information is the same is to take the information in each case from the same source file. At its simplest level, where the common information is in a few, largish blocks, the way to do this is to put each block of the common information in a file of its own and imbed it in the multiple output documents that use it.

IBMIDDoc allows you to reuse elements and information defined in entities; see "File and text entities". You can also reuse information from within the document, see "Reusing elements from an object library" on page 191.

File and text entities

Entities can be used to retrieve document fragments. An entity is any information that is referred to as a unit from a document. All entities are declared at the beginning of a document. Entities cannot be redefined within a document. Entity names are also case sensitive. Thus, `product`, `PRODUCT`, and `Product` refer to different entities.

There are two different types of entities for holding reusable information: file entities and text entities.

File entities
: These are sometimes referred to as external entities or imbeds. These are files of markup and text to include in the document. They can be a large as a chapter (or even larger), or as small as a word (though this will drive translation centers nuts). The declaration points at the file. We recommended that these entities contain complete elements.

| When you save your file entity, give it a meaningful file name. The file
| extension needs to be IDE (or ide), to ensure the ID Workbench processes
| properly recognize the file. Use only letters and numbers in the file name;
| we recommend starting the file name with a letter. Do not include special
| characters in the file name (such as +, –, %, and so forth).

Text entities
: These are sometimes referred to as symbols or internal entities. These entity declarations include the replacement text. A text entity can specify up to 2400 characters of information and markup.

| The content should be a noun string. You should not mix verbs and nouns
| in a text entity, because this will make that entity non-translatable. If you
| do this, the translation centers need to split apart the verbs from the
| nouns. If you need to do this, make an entire sentence an entity.

To include a text or file entity, use an entity reference. An entity reference requests the entity data to be processed at the place where the reference occurs. Any entity can be referenced in this way, but they must be valid in the context in which they are referenced.

The following example shows how text and file entities are defined and used. The entities product, PRODUCT, and Prod are all text entities; the preface is a file entity.

```
<!ENTITY product "ABC Pgm.">
<!ENTITY PRODUCT "PQR Component.">
<!ENTITY Prod "XYZ Pgm">
<!ENTITY preface SYSTEM "xyz10pre.ide">
 ...
This book teaches you how to use &product;.
 ...
&preface;
 ...
<D>Using the &PRODUCT; of the &product;
 ...
<D>Using &product; with &Prod;
```

Special characters

Sometimes you need to specify characters that can be printed on the printer but cannot be typed at your keyboard. An example of that would be the bullet, which looks like this: •

No matter how hard you look, you can’t find a key on your keyboard with one of those on it (unless you have a special keyboard). All symbols are entered the same way: an ampersand (&), followed by the symbol name, followed by a semicolon. So our symbol for the bullet, which is named “bul”, would look like this:

```
&bul;
```

The IBMIDDoc DTD has several symbols for characters you cannot type. These are called character entities. The only SGML-sensitive characters are less-than (<) and ampersand (&) These, when typed in the editor, are automatically converted to `&lt;` and `&amp;` Other characters can be typed if you have them on your keyboard; others you cannot type directly and will need to be entered as character entities.

Table 18 shows the character entities defined in IBMIDDoc.

Table 18. IBMIDDoc Character Entities. All characters are supported in the hardcopy processes. For the online XHTML, HTML, BookManager, and other processes, please check your output for proper appearance and the process logs for messages.

Symbol	Appearance	Description
aa	á	a acute
Aa	Á	A acute
ac	â	a circumflex
Ac	Â	A circumflex
acute	´	accent acute
ae	ä	a umlaut
Ae	Ä	A umlaut
aelig	æ	ae ligature
AElig	Æ	AE ligature
ag	à	a grave
Ag	À	A grave
aleph	ℵ	aleph
all	∀	all
alpha	α	alpha
Alpha	A	Alpha
amp	&	ampersand
and	∧	and symbol

Table 18. IBMIDDoc Character Entities (continued). All characters are supported in the hardcopy processes. For the online XHTML, HTML, BookManager, and other processes, please check your output for proper appearance and the process logs for messages.

Symbol	Appearance	Description
angle	∠	angle
angstrom	Å	angstrom
ao	å	a overcircle
Ao	Å	A overcircle
apos	ʼ	apostrophe
app	≈	approximately
approx	≈	approximately
approxid	≋	approximately identical
arc	∢	arc
asterisk	*	asterisk
at	ã	a tilde
At	Ã	A tilde
atsign	@	at sign
aus	ª	underscored a
ballot	□	ballot box
because	∵	because
beta	β	beta
Beta	Β	Beta
bin	B'	binary
blank	ƀ	blank (b with slash)
box	□	ballot box
BOX	■	solid box
BOXBOT	▄	solid box bottom half
BOXLEFT	▌	solid box left half
BOXRIGHT	▐	solid box right half
BOXTOP	▀	solid box top half
box12	▒	shaded box 1/2 dots
box14	░	shaded box 1/4 dots
box34	▓	shaded box 3/4 dots
bs		backspace
bsl	\	back slash
bslash	\	back slash
bul	•	bullet
bullet	•	bullet
bxas	┴	box ascender
bxbj	┴	box ascender
bxcj	┼	box cross
bxcr	┼	box cross
bxde	┬	box descender
bxh	─	box horizontal
bxle	├	box left junction
bxlj	├	box left junction
bxll	└	box lower-left
bxlr	┘	box lower-right
bxri	┤	box right junction
bxrj	┤	box right junction
bxtj	┬	box descender
bxul	┌	box upper-left
bxur	┐	box upper-right
bxv	│	box vertical
bx0012	╕	ASCII code 184

Table 18. IBMIDDoc Character Entities (continued). All characters are supported in the hardcopy processes. For the online XHTML, HTML, BookManager, and other processes, please check your output for proper appearance and the process logs for messages.

Symbol	Appearance	Description
bx0021	╖	ASCII code 183
bx0022	╗	ASCII code 187
bx0120	╓	ASCII code 214
bx0121	╥	ASCII code 210
bx0202	═	ASCII code 205
bx0210	╒	ASCII code 213
bx0212	╤	ASCII code 209
bx0220	╔	ASCII code 201
bx0222	╦	ASCII code 203
bx1002	╛	ASCII code 190
bx1012	╡	ASCII code 181
bx1200	╘	ASCII code 212
bx1202	╧	ASCII code 207
bx1210	╞	ASCII code 198
bx1212	╪	ASCII code 216
bx2001	╜	ASCII code 189
bx2002	╝	ASCII code 188
bx2020	║	ASCII code 186
bx2021	╢	ASCII code 182
bx2022	╣	ASCII code 185
bx2100	╙	ASCII code 211
bx2101	╨	ASCII code 208
bx2120	╟	ASCII code 199
bx2121	╫	ASCII code 215
bx2200	╚	ASCII code 200
bx2202	╩	ASCII code 202
bx2220	╠	ASCII code 204
bx2222	╬	ASCII code 206
caret	^	caret
cc	ç	c cedilla
Cc	Ç	C cedilla
cdot	⊙	circled dot
cdq	”	close double quote
cdqf	»	French close double quote
cdqg	”	German close double quote
cedilla	¸	cedilla
cent	¢	cent
cequal	⊜	circled equals
char	'	character
check	✓	checkmark
chi	χ	chi
Chi	X	Chi
circ	○	circle
circle	○	circle
CLUB	♣	club solid
cminus	⊖	circled minus
colon	:	colon
comma	,	comma
concat	∥	concatenate
congruent	≃	congruent
cont		continuation character

Table 18. IBMIDDoc Character Entities (continued). All characters are supported in the hardcopy processes. For the online XHTML, HTML, BookManager, and other processes, please check your output for proper appearance and the process logs for messages.

Symbol	Appearance	Description
contains	⊃	contains as a subset
copr	©	copyright
copyr	©	copyright
cplus	⊕	circled plus
csq	’	close single quote
csqg	’	German close single quote
ctimes	⊗	circled times
currency	¤	currency international
cursor	▬	fat cursor
dagger	†	dagger
dahead	▾	down arrowhead
darrow	↓	down arrow
date	October 30, 2001	date
dblldag	‡	double dagger
dbls	§	double S
dblus	‗	underscore double
dblxclam	‼	double exclamation point
dblxclm	‼	double exclamation point
decrease	↘	decrease
def	::=	definition/defined as
deg	°	degree
degree	°	degree
del	∇	del
delta	δ	delta
Delta	Δ	Delta
determines	↦	determines
diam	◇	diamond wide
diamond	◊	diamond
DIAMOND	♦	diamond solid
div	÷	divide
divide	÷	divide
divslash	∕	division slash
dollar	$	dollar
dot	·	dot
dotdot	..	double dot
doubleC	ℂ	double C
doubleN	ℕ	double N
doubleP	ℙ	double P
doubleQ	ℚ	double Q
doubleR	ℝ	double R
doubleZ	ℤ	double Z
Dstroke	Đ	Eth or D stroke
ea	é	e acute
Ea	É	E acute
ebin	'	binary end
ec	ê	e circumflex
Ec	Ê	E circumflex
echar	'	character end
ee	ë	e umlaut
Ee	Ë	E umlaut
eg	è	egrave

Table 18. IBMIDDoc Character Entities (continued). All characters are supported in the hardcopy processes. For the online XHTML, HTML, BookManager, and other processes, please check your output for proper appearance and the process logs for messages.

Symbol	Appearance	Description
Eg	È	E grave
egml	.	gml end tag delimiter
ehex	'	end quoted hex string
ellip	...	ellipsis
ellipsis	...	ellipsis
emdash	—	em dash
endash	–	en dash, dash
epsilon	ϵ	epsilon
Epsilon	Ε	Epsilon
eq	=	equals
eqsym	=	equals
equals	=	equals
eqv	~	equivalent
eserver_logo	@server	e(logo)server
eserver_logo_TM	@server	e(logo)server, trademarked
eta	η	eta
Eta	Η	Eta
eth	ð	eth, Icelandic small
Eth	Ð	eth, Icelandic capital
euler	Ɛ	Eulers
euro	€	Either Euro glyph or EUR
eurochar	€	Either Euro glyph or E
eurotext	EUR	Always EUR
exists	∃	exists
FACE	☻	face solid
face	☺	face
factorial	!	factorial
female	♀	female symbol
ff	ﬀ	ff ligature
ffi	ﬃ	ffi ligature
ffl	ﬄ	ffl ligature
fi	ﬁ	fi ligature
finespace		finespace
fl	ﬂ	fl ligature
florin	ƒ	florin
fnof	ƒ	function of
frac12	½	one half
frac14	¼	one quarter
frac18	⅛	one eighth
frac34	¾	three fourths
frac38	⅜	three eighths
frac58	⅝	five eighths
frac78	⅞	seven eighths
gamma	γ	gamma
Gamma	Γ	Gamma
ge	≥	greater than or equal to
gerank	≽	greater than or equal rank
gesym	≥	greater than or equal to
gml	:	gml delimiter
grave	`	accent grave
gt	>	greater than

Table 18. IBMIDDoc Character Entities (continued). All characters are supported in the hardcopy processes. For the online XHTML, HTML, BookManager, and other processes, please check your output for proper appearance and the process logs for messages.

Symbol	Appearance	Description
gtequiv	≳	greater than or equivilent
gtgt	⋙	much greater than
gtlt	≷	greater than or less than
gtrank	≻	greater than rank
gtsym	>	greater than
hamilton	ℋ	hamiltonian
hat	^	hat
hbar	ℏ	h bar
HEART	♥	heart solid
hex	X'	hex
house	⌂	house
hyphen	-	hyphen
ia	í	i acute
Ia	Í	I acute
ic	î	i circumflex
Ic	Î	I circumflex
Icap	I	I capital character
identical	≡	identical
idotless	ı	i dotless
ie	ï	i umlaut
Ie	Ï	I umlaut
iff	⇔	if and only if
ig	ì	i grave
Ig	Ì	I grave
ij	ĳ	ij ligature
increase	↗	increase
infinity	∞	infinity
intbot	⌡	integral bottom half
integral	∫	integral
intersect	∩	intersection of sets
inttop	⌠	integral top half
inve	¡	inverted exclamation
invellip	⋮	indented vertical ellipsis
invq	¿	inverted question mark
iota	ι	iota
Iota	I	Iota
isubset	⊆	improper subset
isuperset	⊇	improper superset
join	∪	join
kappa	κ	kappa
Kappa	K	Kappa
lahead	◄	left arrowhead
lambda	λ	lambda
Lambda	Λ	Lambda
larrow	←	left arrow
lbarb	↼	left barb
lbrace	{	left brace
lbracket	[	left bracket
lbrc	{	left brace
lbrk	[	left bracket

Table 18. IBMIDDoc Character Entities (continued). All characters are supported in the hardcopy processes. For the online XHTML, HTML, BookManager, and other processes, please check your output for proper appearance and the process logs for messages.

Symbol	**Appearance**	**Description**
lbullet	●	large bullet
ldarrow	⇐	left double arrow
le	≤	less than or equal to
lerank	≼	less than or equal rank
less	≤	less than or equivilent
lesym	≤	less than or equal to
liter	ℓ	liter
lmultdot	•	multiply dot large
lnot	¬	logical not
lnotrev	⌐	backward logical not
lnotusd		upside down not
lor	|	logical or
loz	⌑	lozenge
lozenge	⌑	lozenge
lpar	(	left parenthesis
lparen	(	left parenthesis
lrarrow	⇔	left-right arrow
Lsterling	£	pound sterling
lt	<	less than
ltequiv	≲	ltequiv
ltlt	≪	much less than
ltrank	≺	less than rank
ltsym	<	less than
male	♂	male symbol
mathast	*	mathematics asterisk
mdash	—	em dash
meet	∩	meet
memberof	∈	member of
minus	−	minus operation
minusop	−	minus operation
mp	∓	minus-plus
mu	µ	mu
Mu	M	Mu
mult	×	multiply
ndash	–	en dash, dash
ne	≠	not equal to
nearly	≈	nearly equal
nesym	≠	not equal to
nexists	∄	not existant
nidentical	≢	not identical
nisubset	⊈	not improper subset
nisuperset	⊉	not improper superset
nlerank	⋠	not less or equal rank
nltrank	⊀	not less than rank
nmemberof	∉	not a member of
nnearly	≉	not nearly equal
note1616	♬	pair of 16th notes
note18	♪	eighth note
notsym	¬	not symbol
nsubset	⊆	not a subset
nsuperset	⊇	not a superset

Table 18. IBMIDDoc Character Entities (continued). All characters are supported in the hardcopy processes. For the online XHTML, HTML, BookManager, and other processes, please check your output for proper appearance and the process logs for messages.

Symbol	Appearance	Description
nt	ñ	n tilde
Nt	Ñ	N tilde
nu	ν	nu
Nu	Ν	Nu
numsign	#	number sign
oa	ó	o acute
Oa	Ó	O acute
oc	ô	o circumflex
Oc	Ô	O circumflex
odq	“	open double quote
odqf	«	French open double quote
odqg	„	German open double quote
oe	ö	o umlaut
Oe	Ö	O umlaut
oelig	œ	oe ligature
OElig	Œ	OE ligature
og	ò	o grave
Og	Ò	O grave
omega	ω	omega
Omega	Ω	Omega
omicron	ο	omicron
Omicron	Ο	Omicron
or	∨	or symbol
os	ø	o slash
Os	Ø	O slash
osq	‘	open single quote
osqg	‚	German open single quote
ot	õ	o tilde
Ot	Õ	O tilde
ous	º	underscored o
overline	‾	overline
par	¶	paragraph
parallel	‖	parallel
partial	∂	partial
per	.	period (starter set)
percent	%	percent
period	.	period
perpend	⊥	perpendicular
peseta	₧	peseta
phi	ϕ	phi
Phi	Φ	Phi
pi	π	pi
Pi	Π	Pi
planck	ℏ	h bar
plus	+	plus
plusend	+	plus at end of line
plusmin	±	plus-minus
plusop	+	plus operation
pm	±	plus-minus
prime	′	prime
product	∏	product

Table 18. IBMIDDoc Character Entities (continued). All characters are supported in the hardcopy processes. For the online XHTML, HTML, BookManager, and other processes, please check your output for proper appearance and the process logs for messages.

Symbol	Appearance	Description
proportion	∝	proportion
psi	ψ	psi
Psi	Ψ	Psi
quest	?	question mark
rahead	►	right arrowhead
rarrow	→	right arrow
ratio	:	ratio
rbarb	⇀	right barb
rbl		required blank
rbrace	}	right brace
rbracket	]	right bracket
rbrc	}	right brace
rbrk	]	right bracket
rdarrow	⇒	right double arrow
regtm	®	registered trademark
revbul	◘	reverse bullet
revcir	◙	reverse circle
rho	ρ	rho
Rho	Ρ	Rho
riemann	ℛ	riemann integral
rpar	)	right parenthesis
rparen	)	right parenthesis
rprime	`	right prime
Rx	℞	physician Rx
scriptI	ℐ	script I
scriptl	ℓ	liter
sdq	"	straight double quote
sect	§	double S
section	§	double S
semi	;	semicolon
shiftin	␏	double byte shift in
shiftout	␎	double byte shift out
sigma	σ	sigma
Sigma	Σ	Sigma
similar	~	similar
slash	/	slash right
slr	/	slash right
smultdot	·	muliply dot small
SPADE	♠	spade solid
splitvbar	¦	split veritical bar
sqbul	▪	square bullet
sqbullet	▪	square bullet
sqrt	√	square root
ss	ß	German es-zet
ssbl		syntax significant blank
ssq	'	straight single quote
STAR	★	star solid
sublpar	$_($	subscript left parenthesis
subminus	$_-$	subscript minus
subplus	$_+$	subscript plus
subrpar	$_)$	subscript right parenthesis

Table 18. IBMIDDoc Character Entities (continued). All characters are supported in the hardcopy processes. For the online XHTML, HTML, BookManager, and other processes, please check your output for proper appearance and the process logs for messages.

Symbol	Appearance	Description
subset	⊂	subset of, included in
sub0	$_0$	subscript 0
sub1	$_1$	subscript 1
sub2	$_2$	subscript 2
sub3	$_3$	subscript 3
sub4	$_4$	subscript 4
sub5	$_5$	subscript 5
sub6	$_6$	subscript 6
sub7	$_7$	subscript 7
sub8	$_8$	subscript 8
sub9	$_9$	subscript 9
suchthat	∍	such that
sum	∑	sum
sun	☼	sun
superset	⊃	superset
suplpar	$^($	superscript left parenthesis
supminus	$^-$	superscript minus
supn	^{n}	superscript n
supplus	$^+$	superscript plus
suprpar	$^)$	superscript right parenthesis
sup0	^{0}	superscript 0
sup1	^{1}	superscript 1
sup2	^{2}	superscript 2
sup3	^{3}	superscript 3
sup4	^{4}	superscript 4
sup5	^{5}	superscript 5
sup6	^{6}	superscript 6
sup7	^{7}	superscript 7
sup8	^{8}	superscript 8
sup9	^{9}	superscript 9
tab		tab
tau	τ	tau
Tau	T	Tau
telephone	☏	telephone
TELEPHONE	☎	telephone solid
therefore	∴	therefore
theta	θ	theta
Theta	Θ	Theta
thorn	þ	thorn, Icelandic small
Thorn	Þ	Thorn, Icelandic capital
tilde	~	tilde
time	9:35 a.m.	time
times	×	multiply
tm	™	trademark
TRIANGLE	▲	triangle solid
triangle	△	triangle
ua	ú	u acute
Ua	Ú	U acute
uahead	▴	up arrowhead
uarrow	↑	up arrow
uc	û	u circumflex

Table 18. IBMIDDoc Character Entities (continued). All characters are supported in the hardcopy processes. For the online XHTML, HTML, BookManager, and other processes, please check your output for proper appearance and the process logs for messages.

Symbol	Appearance	Description
Uc	Û	U circumflex
udarrow	⇕	up-down arrow
udarrowus	↨	up/down arrow/underscore
ue	ü	u umlaut
Ue	Ü	U umlaut
ug	ù	u grave
Ug	Ù	U grave
ulbarb	↿	up lef barb
umlaut	¨	umlaut
union	∪	union of 2 sets
upsilon	υ	upsilon
Upsilon	ϒ	Upsilon
urbarb	↾	up right barb
us	_	underscore
usec	µ	micro second
vardelta	∂	delta (variation)
varphi	φ	phi (variation)
varsigma	ς	sigma (variation)
vartheta	ϑ	theta (variation)
vbar	ǀ	vertical bar
vector	⇀	vector
vellip	⋮	vertical ellipsis
weierstr	℘	weierstrass elliptic
won		won (Korean currency)
xclam	!	exclamation point
xclm	!	exclamation point
xi	ξ	xi
Xi	Ξ	Xi
ya	ý	y acute
Ya	Ý	Y acute
ye	ÿ	y umlaut
Ye	Ÿ	Y umlaut
yen	¥	yen
zero	0	zero slashed
zeta	ζ	zeta
Zeta	Z	Zeta
The following entities define the character graphic symbols:		
Ad	↓	arrow down
Al	◂	arrow left
Ar	▸	arrow right
Au	↑	arrow up
Eb	│	end of line, bottom
El	–	end of line, left
Er	–	end of line, right
Et	│	end of line, top
Ju	┼	line junction
Lh	–	line horizontal
Ll	└	lower left corner
Lr	┘	lower right corner
Lv	│	line vertical

Table 18. IBMIDDoc Character Entities (continued). All characters are supported in the hardcopy processes. For the online XHTML, HTML, BookManager, and other processes, please check your output for proper appearance and the process logs for messages.

Symbol	Appearance	Description
Td	┬	T char, bar down
Tl	┤	T char, bar left
Tr	├	T char, bar right
Tu	┴	T char, bar up
Ul	┌	upper left corner
Ur	┐	upper right corner
The following entities make description of the SGML syntax easier to show:		
stago	<	start tag open
etago	</	end tag open
tagc	>	tag close
mdo	<!	markup declaration open
mdc	>	markup declaration close
pio	<?	processing instruc. open
pic	>	processing instruc; close
pero	%	parm entity ref; open
ero	&	entity reference open
erc	;	entity reference close
dso	[	declaration subset open
dsc	]	declaration subset close
msc	]]>	marked section close
lit	"	literal delimiter
lita	'	alternate literal delimiter
The following entities define typographic quote symbols which are used because the Q element has implied citation semantics:		
ctq	”	close typographic quote
otq	“	open typographic quote
The following entities can be used by translation centers for hyphenation:		
shy		soft hyphen

Reusing elements from an object library

Elements that are to be reused many times throughout a document can be defined in an object library. Object libraries are an alternative to using file or text entities. You create object libraries by using the OBJLIB element in the prolog. You place the elements and the content you want to reuse in that object library. The elements in the object library must have an ID.

To use an element from the object library, use the CONLOC attribute to refer to that element. The element in the object library must be the same as the element with the CONLOC attribute. That is, P tags refer to P tags; LI tags refer to LI tags;a PBLK tag cannot refer to a P tag.

The following example shows a small object library and two references to elements in the library. This object library contains an introductory paragraph for service needs. It also contains an ordered list of things that must be done if service is required.

```
<prolog>
⋮
<objlib>
<objlibbody>
<p id="para1">If your system stops
```

```
working, follow these instructions:</p>
<ol id="list2">
<li>Note the system code displayed on the front of
the unit.</li>
<li>Unplug the unit.</li>
<li>Contact your service representative.</li>
</ol>
</objlibbody>
</objlib>
⋮
</prolog>
⋮
<d>
<dprolog><titleblk>
<title>If you need service</title>
</titleblk></dprolog>
<dbody>
<p conloc="para1">
<ol conloc="list2">
</dbody></d>
```

When the P element with the CONLOC attribute of “para1” is processed, the content of the P element in the OBJLIB with the “para1” attribute is used. This markup portion:

```
<p conloc="para1">
```

Is the same as specifying this markup:

```
<p>If your system stops
working, follow these instructions:</p>
```

The content of an element defined in an object library is used only if you refer to that element in the document content.

An element defined in an object library can be referred to only from within the document containing the object library. If you have information that will be re-used by other documents, the object library can be declared as a file entity and imbedded in each document. This level of reuse allows much more flexibility and function for reuse across documents.

Migration Note

ObjLib can also be used as, in the Bookmaster paradigm, a DVCF side file that uses the include macro.

The element with the ID must be a direct child of the ObjLibBody tag. You cannot use a CONLOC to refer to an element nested inside something else in the ObjLibBody tag. For example, referencing the LI element “unplug”in the next example is **not correct**:

```
<objlib>
<objlibbody>
<ol id="list2">
<li>Note the system code displayed on the front of
the unit.</li>
<li id="unplug">Unplug the unit.</li>
<li>Contact your service representative.</li>
</ol>
</objlibbody>
</objlib>
```

To correct the example, you need move the LI outside the list, and include an LI with a CONLOC. The items reused within an object library must be defined before they are referenced, so the LI is before the list.

```
<objlib>
<objlibbody>
<li id="unplug">Unplug the unit.</li>
<ol id="list2">
<li>Note the system code displayed on the front of
the unit.</li>
<li conloc="unplug">
<li>Contact your service representative.</li>
</ol>
</objlibbody>
</objlib>
```

If you want to reuse content of a part of a list, containing the content you wish to reuse within an LIBlk element makes it easy to reference the LIBLk using the CONLOC attribute.

You can have several divisions that get reused by using the DBLK tag to contain those divisions.

Reusing attributes in the CONLOC reference

Starting with IDWB release 3.4, patch IDWXF036: The attributes for an item in an object library are now passed through to the reference. For example, you have a list item with a revision ID:

```
<objlib>
<objlibbody>
<li id="renew" rev="rel34a">Renew your subscription</li>
</objlibbody>
</objlib>
```

You can refer to the list item, and the REV attribute is carried along. For example:

```
<li conloc="renew">
```

is now the same as this:

```
<li rev="rel34a">Renew your subscription</li>
```

Before patch IDWXF036, you would have only gotten the text, the REV attribute would be ignored:

```
<li>Renew your subscription</li>
```

If the item in the object library and its reference have the same attributes, the value on the CONLOC reference wins.

Cross-referencing items that use CONLOC

Now, for every solution, there is a problem.[7] If you want to cross-reference an item with a CONLOC, you need to add the ID to the tag with the CONLOC. For example, you want to reuse a division, plus cross-reference to it. You cannot cross-reference to “service”. You need to add unique IDs to each division. This applies to items who’s parent is the ObjLibBody tag.

```
<objlib>
<objlibbody>
<d id="service">
<dprolog><titleblk>
```

7. Yes, I wrote that correctly.

```
<title>If you need service</title>
</titleblk></dprolog>
<dbody>
<p>If you need service...
</dbody></d></objlibbody>
</objlib>
 ...
<d conloc="service" id="abc">
 ...
<d conloc="service" id="def">
 ...
<xref refid="abc">
 ...
<xref refid="def">
```

Any ID within an element who’s grandparent is the ObjLibBody tag will, when re-used, have a reference to the first use of that item.

Chapter 19. Conditionally including information

Where multiple output documents contain common information, or a single document repeats the same information, the only way to ensure that the common information is the same is to take the information in each case from the same source file. At its simplest level, where the common information is in a few, largish blocks, the way to do this is to put each block of the common information in a file of its own and imbed it in the multiple output documents that use it.

However, this process becomes cumbersome and inefficient where the common information is strewn throughout the document, or, alternatively, where the differences for the multiple output documents are scattered through the common information. This approach also has the drawback that reviewers of the documents must do redundant reviewing of the common information. For these reasons, we have property-based retrieval. See "Property-Based Retrieval".

There are also times when you need alternative text under different conditions. You can specify the modification text (the insertion or replacement) either in line in the source file or out of line in an object library. See "Retrieval alternatives" on page 198.

Property-Based Retrieval

Property-based retrieval allows you to structure documents such that different versions can be produced based on a property value file (that is, a set of one or more conditions). With property-based retrieval, you can:

- Insert, delete, or replace text based on conditions you specify.
- Specify conditions that are simple or complex.
- Pass the conditions at run time or inside the document itself.

In IBMIDDoc, conditional processing is done by evaluating certain attribute values on elements to determine whether or not those elements should be processed. The attributes are called ***property attributes*** because they define properties of elements.

Using the Props attribute to set text conditions

The Props attribute on a tag, when true, causes that content to be processed and appear. The attribute can contain a simple condition that is either true or false, or a complex condition of Boolean operators. The attribute's value, the specification, can be any valid SGML name, but should be a word or phrase that is clear and meaningful to those who are writing or editing the information. The specification may be a version, a software code name, or a hardware platform; for example, v4r5, win32, or PC, respectively.

It is up to those working on a product or group of products to choose consistent terminology for assigning occurrences of the Props attribute. The names and meaning should be documented and available to all writers and editors involved in the development effort to ensure consistency.

For example, each these items will format when the conditions v4r5, win32, and PC are all true:

```
<ul>
<li props="v4r5">The version 4, release 5 level has
special stuff.</li>
<li props="win32">This software release has more special
stuff.</li>
<li props="PC">Is the PC going to be replaced?</li>
</ul>
```

Translation Center Considerations: Conditional text can be a powerful feature, but you need to use care because of translation considerations. If you have a condition within a sentence, ensure it is a noun string. Do not combine a noun and a verb in the same conditional phrase. Not every language has sentence construction like English (or whatever language you are writing in). If you need to combine nouns and verns in the same conditionap phrase, make an entire sentence conditional.

If you have a condition within a sentence, ensure your sentence makes sense with all possible logic conditions. Here are some examples:

- Here's a good example:

  ```
  The <ph props="os2">OS/2</ph><ph props="#NOT os2">Windows</ph> operating
  system runs on PCs.
  ```

  This way you always get a complete sentence, as in this when "os2" is true:

  > The OS/2 operating system runs on PCs.

  And this when "os2" is false:

  > The Windows operating system runs on PCs.

- In this next example — you get an incomplete sentence when both conditions are false:

  ```
  When <ph props="equine">horses gallop</ph><ph props="canine">dogs run</ph>
  down the track, you will know the race has begun.
  ```

  When both "equine" and "canine" are false, the sentence becomes:

  > When down the track, you will know the race has begun.

Referring to something that is conditional: Imagine you have two paragraphs (or two somethings) that have an either condition. Now you want to refer to the paragraph. You give each of them the same ID, and then the editor and the IDWB processes complain because the IDs appear twice. As in this example:

```
<p id="cats" props="cats">Cats are nice.</p>
<p id="cats" props="#NOT cats">Cats are not nice.</p>
...
See <xref refid="cats" xreftext="cat feelings"> for how I feel about cats.
```

We have two paragraphs; they are mutually exclusive, but because they have the same ID, SGML rules complain about the duplicate IDs.

So what to do? Put the two paragraphs in a paragraph block (Pblk), and put the ID on the Pblk tag! For example:

```
<pblk id="cats">
<p props="cats">Cats are nice.</p>
<p props="#NOT cats">Cats are not nice.</p>
</pblk>
...
See <xref refid="cats" xreftext="cat feelings"> for how I feel about cats.
```

| We now have one ID — no more duplicate ID problem. (But now you might have
| a problem with cat owners).

Setting the properties to true or false

The properties are initially assumed to be false. When you process your document, messages indicate that when a condition is found, false is assumed.

To set a property value, you can do the following:

- Use a VAL file to set the properties to true or false. This is described in the *ID Workbench Getting Started and User's Guide* and in the online help for ID Workbench. See the **Transform** tab in the processing option displays.
- Use a PropDesc tag in the document's prolog; within the PropDefs section.

Here is an example showing how to set the properties “v4r5” and “PC” to true and false, respoctively.

```
<prolog>
 ...
<propdefs>
 ...
<propdesc propname="v4r5" default="true">
<desc>Version 4, release 5</desc>
</propdesc>
<propdesc propname="pc" default="false">
<desc>PC platform information.</desc>
</propdesc>
 ...
</propdefs>
 ...
</prolog>
```

Specifying boolean properties

Sometimes you need to specify fancier conditions, called complex conditions. Suppose we have two conditions, A and B. To have an action (insert or delete) occur:

- When (and only when) both conditions are true, enter:
  ```
  a #AND b
  ```
- When either one or both of the conditions is true, enter:
  ```
  a #OR b
  ```
- When a condition is not true, enter:
  ```
  #NOT a
  ```

You might have a situation where you want one sentence for one condition, and different sentence for the opposite condition. Instead of having two properties and setting one to true and the other to false; you can just have one property and use the #NOT operator. For example:

```
<ul>
<li>Always use this item</li>
<li props="PC">This is a PC-only item</li>
<li props="#NOT PC">This item is for everything EXCEPT PCs</li>
</ul>
```

The order of precedence in evaluation is:

1. specifications inside parentheses are evaluated first
2. #NOT specifications are evaluated next
3. Finally, #AND and #OR operators are evaluated from left to right

These three functions, #AND, #OR, and #NOT, can be strung out to the point where only a computer could figure out what to do. See Table 19 for a set of sample conditions and the results with different logic groupings.

Table 19. Property Truth Table. T means true; blank indicates false.

These conditions			**Yield these complex conditions:**						
A	**B**	**C**	**A #and B**	**A #or B**	**A #and #not B**	**A #and B #or C**	**A #or B #and C**	**A #and #not B #or C**	**A #or #not B**
T	T	T	T	T		T	T	T	T
T	T	F	T	T		T	T		T
T	F	T		T	T	T	T	T	T
T	F	F		T	T		T	T	T
F	T	T		T		T	T	T	
F	T	F		T					
F	F	T				T		T	T
F	F	F							T

This example shows that the second paragraph applies to MVS in both versions of the product, and to VM in only the first version of the product.

```
<p PROPS="(VM #or MVS) #and (V1 #or V2)">
This paragraph applies to both versions and operating systems.
</p>
<p PROPS="MVS #or (VM #and V1)">
This paragraph applies to MVS or version 1 on VM.
</p>
<p PROPS="VM #and V2">
This paragraph applies only to version 2 on on VM.
</p>
```

Retrieval alternatives

Sometimes you need alternative text for a condition. The RETALTS attribute can point to one or more retrieval alternatives for the text in your document.

You need to create your alternative text in an object library in the prolog of your document. The main element will need an ID. See "Reusing elements from an object library" on page 191 for information about creating object libraries.

To have the alternative text be used:

- The main element must have a false Props attribute, and a Retalts attribute with one or more IDs.
- The Retalts attribute points at the ID of a replacement element found in the object library. The elements must be the same type. The first true element is used for the alternative text.

The following example illustrates this type of conditional processing.

```
<prolog>
 ...
<objlib>
<objlibbody>
<p id="v1intro" props="v1">This is an introduction for version 1...</p>
<p id="v2intro" props="v2">This is an introduction for version 2...</p>
<p id="v3intro">This is an introduction for version 3...</p>
```

```
</objlibbody>
</objlib>
 ...
</prolog>
 ...
<p props="x" retalts="v1intro v2intro v3intro">This
is an introduction...</p>
```

The paragraphs will print under these conditions:

- When “x” is true, this prints:

  This is an introduction...
- When “x” is false and “v1” is true, this prints:

  This is an introduction for version 1...
- When “x” and “v1” are false and “v2” is true, this prints:

  This is an introduction for version 2...
- When “x”, “v1”, and “v2” are false, this prints:

  This is an introduction for version 3...

Using Marked Sections

Marked sections provide two key functions:

1. they allow conditional inclusion or exclusion of material and
2. they control SGML delimiter recognition for documenting SGML and markup as well as documenting other subjects that use SGML markup characters for other purposes.

Marked sections should not be used to provide conditional processing capability. Use the property-based retrieval function instead.

Marked sections have the following format:

```
<![ keyword status area [marked section data]]>
```

keyword status area
: This is specifications that control the function of the marked section. See below for possible values.

marked section data
: data to be treated based on the content of the keyword status area.

Marked sections support two keywords for conditional inclusion and exclusion: IGNORE and INCLUDE. One or both are specified in the keyword status area described above. If both are specified, IGNORE has higher precedence. Here is an example:

```
<![ IGNORE [
This information will be ignored.
]]>

<![ INCLUDE [
This information will be included.
]]>
```

Why would anyone would specify both INCLUDE and IGNORE? The keyword status area may include parameter entity references that allow the author to parameterize these inclusions from the document prolog without changing the document (and marked section keyword status area) content. Examples of this follow the parser recognition control description.

Parameter entities may be used to parameterize the keywords found in a marked section keyword status area. This is particularly useful in conditional processing cases. For example, assuming you have material that is intended for two uses, say reference cards and full language reference, you can encode both in the same document and then change the parameters to include just the material for the output currently desired. Here is an example:

```
<!DOCTYPE IBMIDDOC PUBLIC "-//..." [
<!ENTITY % langrefonly "include" >
<!ENTITY % refcardonly "ignore" >
   ⋮
<P>Material that belongs in both output docs doesn't have any marked
section markup
<![ %langrefonly; [
This is material that goes only in the language reference.
]]>
<![ %refcardonly; [
This is material that goes only in the reference card.
]]>
More material that goes in both.
```

Controlling SGML Delimiter Recognition

There are two keywords to control SGML delimiter recognition:

CDATA
: inhibits the recognition of all markup except the marked section close delimiters ’]]>’

RCDATA
: supports recognition of entity references and the marked section close delimiter.

You use this for including SGML markup examples and for including other material that uses SGML markup delimiter characters in other ways:

```
<![ CDATA [
<P>This is an example paragraph with an example entity reference:
&entref;.
]]>

<![ RCDATA [
<p>This is an example paragraph of IBMIDoc coding with an example symbol
reference: &bkmsym; ]]></p>
```

In the second case, the &amp; SGML entity resolves to an & that gives the correct result in documenting IBMIDoc encoding.

Chapter 20. Property and Class Definitions

This chapter describes how to define element properties, element classes, and properties for element classes. Some of these definitions can also be done with DEF tags; see Chapter 9, “Using definition tags” on page 105.

If the referenced element also has a PROPSRC specification, this reference is followed until the end of the chain of property specification is reached. Property specifications on the referencing element override any properties from referenced or inherited property specifications. The hierarchy of property use is:

1. properties specified on the element,
2. properties specified on an element which is referenced using the CONLOC attribute or properties specified on an element which is referenced using the RETALTS attribute and whose properties were satisfied for this processing run (the CONLOC and RETALTS referenced elements are treated as independent elements and their properties do not interact),
3. properties from a PROPSRC referenced element, however long the reference chain may be,
4. properties from ClassDef elements referenced by class.
5. properties specified on a PropDef element without an ID (with or without an ELETYPES attribute)
6. properties from ancestors in the document tree

Defining Element Properties

In IBMIDDoc, you can define properties for an element, such as language, status, and classification. You can define properties directly, by linking to another element, through inheritance, or, for security classification, by implying it from an element’s children. You can also define reusable sets of properties.

Defining Element Properties Directly

Properties can be defined for an element by using the property attributes.

The Propdef tag sets default properties for tags that point at them with propsrc attributes. This allows you to set defaults once, and reference them. When something changes, you only have to change the propdef). The only attribute currently able to be passed from PROPDEF to an element is STYLE. For example; this propdef sets a style of bold for the propname fred:

```
<propdef propname="fred" style="bold">
```

Now, when a tag that supports a bold style uses the fred property, the content will be bold:

```
<ph propsrc="fred">hi there</ph>
```

Values of the same attributes on the tags override the values on the propdef tag. The following tag, because the style attribute is used on the tag itself, overrides the style attribute on the propdef. It will be italic:

```
<ph propsrc="fred" style="italic">hi there</ph>
```

The following shows a propdef that sets a style of bold and a conditional property of aix:

```
<propdef propname="fredaix" style="bold" props="aix">
```

Consider this tag:

```
<ph propsrc="fredaix" props="win">hi there</ph>
```

Because the attributes on the tags override the same attributes on the propdef, the ph tag effectively becomes the following:

```
<ph style="bold" props="win">hi there</ph>
```

This is because the props attribute on the tag overrides the props attribute on the propdef. The style attribute on the propdef is carried through to the tag.

You can leave off the propname and propsrc to get a property default for all the tags. For example, this propdef sets a style of bold for any tag that supports a style of bold:

```
<propdef style="bold">
```

These will print in bold:

```
<ph>hi there</ph>
<term>hello again</term>
```

Additionally, because you can define style overrides on both propdef and classdef, then use them together, the ID Workbench uses a rule determines which will win. The special override rule for style and class when used together is: 1) Style on the tag, then 2) Class on the tag, then 3) Style on Propdef.

The following example shows a paragraph with defined properties of OS/2 V2.1 only.

```
<P props="os2 #AND v21">This paragraph
is used for OS/2 V2.1 only.
```

Properties defined directly take precedence over properties defined by linking or through inheritance. The only exception is the security classification property, which can be implied from an element's children.

Defining Element Properties Using Inheritance

All elements inherit their parent's properties. For example, a paragraph within a division that has a language property value of Spanish will inherit that property and will thus be identified as Spanish as well.

Defining Element Classes

You can use the ClassDef element to define classes of IBMIDDoc elements.

Typically, element classes are used to define specific phrase classes that reflect the product being described. You can enable the various processing functions by using the Phrase elements with the classes you define.

Usually, element classes are defined for an entire collection of documents by someone responsible for designing the information in the collection, such as an information designer or planner. If you are working on information for which element classes have been defined, you do not need to understand how classes are defined. However, you do need to know the class names and the affected elements.

The default IBMIDDoc templates have the following phrase classes defined. They are enabled in the editors and appear properly when used.

Class	Description
IBMGUIControl	Use this for graphical user interface items; such as menu items, pushbuttons, icons, and so forth. Press the **OK** button. `Press the <ph class="IBMGUIControl">OK</ph> button.`
IBMCommand	Use this for command names, APIS names, functions names, and so forth. Run the **Copy** command. `Run the <ph class="IBMCommand">Copy</ph> command.`
IBMEmphasis	Use this for things you want to emphasize. Look *here* please. `Look <ph class="IBMEmphasis">here</ph> please.`
	Use this for file and path names. The file name is system.ini. `The file name is <ph class="IBMfilepath">system.ini </ph>.`

In defining element classes, first determine what classes are needed and decide the class names. Analyze your product to identify what kinds of things you need to write about. Classes should be meaningful and should describe real things or aspects of real things. Classes should not relate purely to processing or presentation effects. Thus, a class of *bold* is probably not meaningful. Usually, class names should be nouns.

Next, define exactly what the classes are so that you understand when and why to use them.

Then, define the element classes using ClassDef elements. ClassDef elements are valid within PropDefs, which is in either the document prolog or a division prolog. If a class applies to an entire document, put the ClassDef element in the document prolog. Use the Sem element within each ClassDef element to describe the class.

Finally, after you define the classes, use the class names with the elements to which they apply to assign element classes in your document.

Use ClassDef to define element classes that are specific to your information. The most common use of ClassDef is to define new phrase classes. For example, in the documentation of a graphical user interface, you may want to define phrase classes for all the different types of user interface elements in order to make your markup more precise and to enable the automatic generation of indexes for print presentation.

You can associate presentation styles and other processing with specific element classes. Do not define classes that are purely presentational, such as Bold or Italic.

Suppose, for example, that you are documenting software that uses two important types of objects, Whatsits and Thingies, that are not accounted for in the base IBMIDDoc language. Whatsits are hardware components, and Thingies are software components. The class names for Whatsit and Thingy objects are “Whatsit” and “Thingy”, respectively. Because the Whatsit and Thingy classes apply to an entire document, they are defined in the prolog.

The following example shows the ClassDef elements that define the Whatsit and Thingy classes and the use of the classes with the Ph element:

```
<PROLOG>
   ⋮
 <PROPDEFS>
  <CLASSDEF CLASSNAME="whatsit">
   <SEM>Identifies whatsit objects.  Whatsits are hardware
        components.
  </CLASSDEF>
  <CLASSDEF CLASSNAME="thingy">
   <SEM>Identifies thingy objects.  Thingies are software
        components.
  </CLASSDEF>
 </PROPDEFS>
   ⋮
</PROLOG>
   ⋮
 <D>Hardware Problems
  <P>Hardware problems are usually caused when
    the <PH CLASS="whatsit">famtoozler</PH>
    gets out of adjustment.
    Readjust it using the <PH CLASS="thingy">famtoozlometer</PH>
    component of the &prodname; analyzer.
```

Chapter 21. Making some things bigger or smaller

Need to change the type size of something? Have we got a deal for you. These topics describe the following:

- "Scaling text up or down"
- "Automatically scaling text for examples and such"
- "Making things page-wide"

You can also scale the number or bullets in lists; see "Scaling list dingbats" on page 36.

Scaling text up or down

You can make some things bigger or smaller. The SCALEPCT attribute on the Table, Fig, Syntax, and MsgList tags allow you to scale the text larger or smaller. Here's a sample of a table with larger type:

Large Type Table	150% normal size.
`<table scalepct="150">`	Easy on the Eyes

Here's a table with smaller type:

Small Type Table	70% normal size.
`<table scalepct="70">`	10 pounds of stuff in a 5 pound sack

The scale value needs to be a positive, whole number that is 1 or greater. There is a limit to the sizes of fonts that are available. Your scale value will be rounded to the closest available font. This currently works for hardcopy formatted hardcopy.

Automatically scaling text for examples and such

For automatic scaling down of wide examples, screens, and character graphics, you can use the LINELENGTH attribute. You specify the width of the widest line, and the formatter automatically scales down the text, if needed, to fit within the current column or page width. Here's an example with a width of 96 characters; the formatter automaticaly scaled the text down to fit across the page:

```
<xmp linelength="96">
This has a width of 95 characters, and the formatter automatically scales the text down for us.
</xmp>
```

This currently works for hardcopy formatted hardcopy.

Making things page-wide

You can also use the PGWIDE attribute on tables, figures, syntax diagrams, and examples to force them to be page wide. PGWIDE=1 causes the item to be page-wide. For example:

```
<xmp pgwide="1">
Here's a really wide example. This could be for a listing of sample programming code.
</xmp>
```

This currently works for hardcopy formatted hardcopy.

Chapter 22. Creating maintenance analysis procedures

IBMIDDoc provides a handy format for writing step-by-step procedures to help isolate the cause of a symptom. These procedures are called Maintenance Analysis Procedures (MAPs). If you are familiar with flowcharts, you know how they lead you through a sometimes complex series of steps by having you make one simple yes-or-no decision at a time. However, flowcharts can be difficult to work with because the flowcharting symbols contain so little space for writing questions, directions, or other text. We solve this space problem while keeping the technique of using simple yes-and-no answers to lead people through their procedures.

Procedures consist of several parts:

- procedure entry (see "Using ProcEntry for Entry Requirements" on page 209)
- procedure steps and commands (see "Using ProcStep and ProcCmnd to Describe Each Step" on page 209)
- decision points (see "Using DecisionPnt for Outcome-Dependent Action Descriptions" on page 210)
- reference keys (see "Using RefKeys to Refer to Labels in a Graphic" on page 210)
- procedure exit (see "Using ProcExit to Complete a Procedure or Sub-Procedure" on page 211)

| The following shows a sample map some father made for caring for his little one.
| Xyvision and Frame2000 currently do not draw vertical rules from steps to link
| points; BookMaster does draw these rules.

MAP 0010: Baby Johnny is crying

Six-month-old Baby Johnny was sleeping peacefully; suddenly he began to cry.

$\boxed{\textbf{001}}$

– Check Johnny's diaper.

Is the diaper wet?
Yes No
Continue at Step 003.

$\boxed{\textbf{002}}$

– Change the diaper.

Johnny was uncomfortable.

$\boxed{\textbf{003}}$

(From step 001)

Is Johnny hungry?
Yes No

$\boxed{\textbf{004}}$

– Rock Johnny to sleep.

Johnny was sleepy.

$\boxed{\textbf{005}}$

Does Johnny have teeth?
Yes No

$\boxed{\textbf{006}}$

– Warm a bottle.
– Feed Johnny.

Johnny needed a bottle.

$\boxed{\textbf{007}}$

Johnny can eat solid food.
Continue at "MAP 0020: The Steak is Frozen" on page 209.

MAP 0020: The Steak is Frozen

$\boxed{001}$

Do you have a microwave oven?
Yes No

$\boxed{002}$

– Johnny can't wait for it to thaw.

Continue at Step 006 on page 208.

$\boxed{003}$

– Thaw the steak.

Using ProcEntry for Entry Requirements

The ProcEntry element contains a description of the entry point to the procedure. It contains the description, and references to any prerequisite or related procedures. Related and prerequisite procedures are referenced by ID using the RELPROCS and PREREQPROCS attributes.

```
<PROCENTRY PREREQPROCS="proca" RELPROCS="proc1 proc2 proc3">
SIX-MONTH-OLD BABY JOHNNY WAS SLEEPING PEACEFULLY;
SUDDENLY HE BEGAN TO CRY.
</PROCENTRY>
```

Using ProcStep and ProcCmnd to Describe Each Step

The ProcStep element contains the actions to take and the expected results of taking the actions. The title for each ProcStep is contained in the required TitleBlk elements. The Desc element contains the description of the action that must be performed.

The ProcCmnd element contains specific instructions that the user must follow in order to complete the step.

```
<proc id="babymap" style="BKM:(STYLE=BASE SEP=INLINE COMPACT)">
<titleblk><title>Baby Johnny is Crying</title></titleblk>
<procentry>Six-month old baby Johnny was sleeping
peacefully. Suddenly he began to cry.</procentry>
<procstep>
<proccmnd>
<desc>Check Johnny's diaper.</desc>
</proccmnd>
<decisionpnt>
<cond>Is the diaper wet?</cond>
<then><procstep><proccmnd>
<desc>Change the diaper.</desc>
</proccmnd><procexit>Johnny was uncomfortable.</procexit>
</procstep>
</then>
<else>
<desc>Continue at <xref refid="hungry">.</desc>
</else>
</decisionpnt>
</procstep><procstep id="hungry">
<decisionpnt>
<cond>Is Johnny hungry?</cond>
```

```
<then><procstep><decisionpnt>
<cond>Does Johnny have teeth?</cond>
<then><procstep><stepnotes><li>Johnny can eat solid
food.</li>
<li>Continue at <xref refid="frozstk"></li>
</stepnotes></procstep>
</then>
<else><procstep id="bottle"><proccmnd>
<desc>Warm a bottle.</desc>
</proccmnd><proccmnd>
<desc>Feed Johnny.</desc>
</proccmnd><procexit>Johnny needed a bottle.</procexit>
</procstep>
</else>
</decisionpnt></procstep>
</then>
<else><procstep><proccmnd>
<desc>Rock Johnny to sleep.</desc>
</proccmnd><procexit>Johnny was sleepy.</procexit>
</procstep>
</else>
</decisionpnt>
</procstep><procstep id="frozstk">
<proccmnd>
<desc>Thaw and broil a steak for Johnny. Include a
baked potato with butter and sour cream.</desc>
</proccmnd>
<procexit>Johnny was really hungry.</procexit>
</procstep></proc>
```

Using DecisionPnt for Outcome-Dependent Action Descriptions

The DecisionPnt element defines one or more condition/action (Then/Else) pairs that describe actions that must be completed under certain conditions.

```
<DECISIONPNT>
 <COND>IS THE DIAPER WET?</COND>
 <THEN>
  <PROCSTEP>
   <PROCCMND>
    <DESC>CHANGE THE DIAPER.</DESC>
   </PROCCMND>
   <PROCEXIT>JOHNNY WAS UNCOMFORTABLE.</PROCEXIT>
  </PROCSTEP>
 </THEN>
 <ELSE>
 <DESC>CONTINUE AT <XREF REFID="HUNGRY">.</DESC>
 </ELSE>
</DECISIONPNT>
```

Using RefKeys to Refer to Labels in a Graphic

The RefKey element contains a reference to a label in a graphic. When processed, this label provides a visual link to a spot in, for example, a graphic containing a chart or table.

```
   ⋮
<LI><P>The current 1995 Sales chart column
 <REFKEY>4</REFKEY>
shows that sales
are up 10%, but operating expenses grew by 13.2%.</P>
</LI>
```

Using ProcExit to Complete a Procedure or Sub-Procedure

The ProcExit element contains the expected result of performing the task, and describes what to do after completing the procedure tasks.

```
<PROCSTEP>
 <PROCCMND>
  <DESC>ROCK JOHNNY TO SLEEP.</DESC>
 </PROCCMND>
 <PROCEXIT>JOHNNY WAS SLEEPY.</PROCEXIT>
</PROCSTEP>
```

Procedure Markup Examples

The examples that follow illustrates the use of procedure elements in IBMIDDoc.

Starting the Procedure

All Proc elements must contain a TitleBlk element, and a Desc element that contains a description of the procedure's purpose. The STYLE attribute on the Proc element in the example that follows specifies a value of STEPLIST

```
<PROC STYLE="steplist" id="proc1">
 <TITLEBLK><TITLE>Installing the ISDN Whantoozler</TITLE></TITLEBLK>
  <DESC>This procedure describes the steps one must perform or follow
        in order to successfully install the ISDN Whantoozler.
  </DESC>
   ⋮
```

Describing the Entry Point for the Procedure

The ProcEntry element contains the description of the entry point for the procedure. In the next portion of the example, the ProcEntry element contains RelProcs and PreReqProcs attributes, which reference related and prerequisite procedures. It also includes a prose description of the entry point for the procedure.

Note that the format in which related and prerequisite procedures are presented is style and processing dependent.

```
   ⋮
 <PROCENTRY PREREQPROCS="proc1a" RELPROCS="proc3 proc2">
  This procedure assumes that you already have your
  ISDN line installed, and that there is no
  thunderstorm activity in the area.
 </PROCENTRY>

   ⋮
```

Entering the Procedure Steps

Each procedure step is contained in a ProcStep element, which contains the title of the step and the step instructions. ProcStep may also contain:

- StepNotes, which allow you to make general notes about the step
- DescisionPnt which contain decision-making information for the step.

```
<PROCSTEP>
<TITLEBLK><TITLE>Pre-Configuring ISDN Whantoozler</TITLE></TITLEBLK>
 <PROCCMND><DESC>Set the 4 DIP switches on the ISDN Whantoozler
 to correspond to the hemispheric location of your ISDN server.</DESC>
 </PROCCMND>
 <STEPNOTES>
  <LI>NA-ISDN Server:  SW1234 ON ON ON  ON</LI>
  <LI>SA-ISDN Server:  SW1234 ON ON ON  OFF</LI>
  <LI>EU-ISDN Server:  SW1234 ON ON OFF OFF</LI>
  <LI>AU-ISDN Server:  SW1234 ON ON OFF ON</LI>
 </STEPNOTES>
 <DECISIONPNT>
  <COND>Do you know your hemispheric location?</COND>
   <THEN>
    <PROCSTEP>
     <PROCCMND><DESC>Continue to step <XREF REFID="nextstep">.</DESC>
   </THEN>
   <ELSE>
     <PROCSTEP>
     <PROCCMND><DESC>Find out the information and retry this step.</DESC>
    </PROCSTEP>
   </ELSE>
 </DECISIONPNT>
</PROCSTEP>
```

Exiting the Procedure

The end of the procedure is described in the ProcExit element. You must include a description. You can also include the RECOVERYPROC attribute.

```
<PROCEXIT ID="pxita" RECOVERYPROC="rc1">
  <P>The ISDN Whantoozler should have installed without problems.
  The machine should have powered up
  successfully.  If so, you may continue to <XREF REFID="ok1">.
  <P>If the machine smoked when you applied power,
  see <XREF REFID="rc1"> for troubleshooting information.
</PROCEXIT>
```

Controlling Procedure Output Styles

Future Enhancement

Control of procedure output styles may be implemented in future release. These are presented for proposals only; they are not presently working. They are not slated for inclusion in any furture release. If you need these sorts of output, please submit a requirement.

The default style for procedures is a MAP style. A typical MAP style output is illustrated in the formatted example that follows.

There is limited HTML and IPF support for Maintenance Analysis Procedures. When outputting to RTF, IPF, or Windows Help, MAPS/PROC become nested divisions. For hardcopy, the output is placed in a “flowchart” type mode.

The three proposed styles are :

Plaintext
results in a procedure with headings as the step numbers, and the step descriptions contained in paragraphs

Steplist
looks like an ordered list with Step 1, Step 2, Step 3, and so forth, as the numbering scheme.

1. This is the first step description.
2. This is the second step description.
3. This is the third step description.

Procedure exit description.

Table presents the procedure information in a table format, as shown in the following example.

Table 20. Test of Prereq and Coreq

Step	Description	Reference Keys
1	This is the first step description. It contains several paragraphs of information. This is the procedure entry. This is some more information about the procedure. And here is even more information.	
2	This is the second step description. This step contains both a decision point and a step notes section. To continue: **IF:** The step worked. **THEN:** Continue to step 3. **IF:** The step did not work. **THEN:** Ensure that all cables are connected, and repeat the step again. **Notes:** 1. This is the first note 2. The second note.	
3	This is the third step description. This step includes explicit reference key elements.	A b c

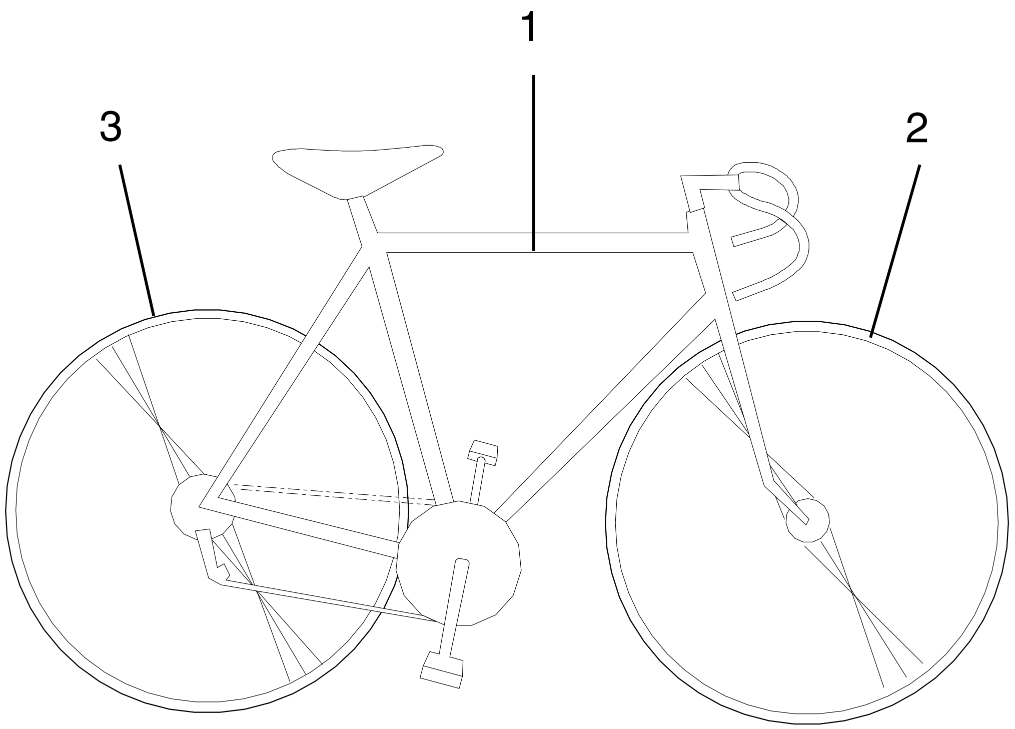
Proc Exit:

Procedure exit description.

Chapter 23. Creating parts catalog lists

A list of parts in a catalog is presented in conjunction with an illustration that shows where the parts fit in an **assembly** (a collection of parts that make up a unit of a machine or other product). The parts (also called **component items**) in the component list are keyed to numbered callouts on the artwork. IBMIDDoc provides an effective way to present both the component list in a parts catalog and the artwork associated with it. Component lists are usually presented in one of two ways: with the artwork on the top of a page followed by the list with the artwork showing the assembly on the left, (even-numbered page), and the list on the right, (odd-numbered page).

Assembly 1: Bicycle



Asm– Index	Part Number	Units	Description
1–1	4563423	1	Bike
–1	1230987	1	• Frame
–2	1238475	1	• Wheel assembly, front
			For detailed breakdown, see "Assembly 2: Wheel, front" on page 219.
–3	1234939	1	• Wheel assembly, rear

Markup source

```
<partasm id="bike" style="bkm:(layout=same)"><title>
Bicycle</title><mmobj><objref obj="bike">
<textalt>Bicycle</textalt>
</mmobj><compl>
<ci idxnum="1" partnum="4563423" upa="1">Bike</ci>
<compl>
<ci idxnum="1" partnum="1230987" upa="1">Frame</ci>
<ci idxnum="2" partnum="1238475" upa="1">Wheel assembly, front</ci>
<compcmt>For detailed breakdown, see <xref refid="wheelxmp">.</compcmt>
<ci idxnum="3" partnum="1234939" upa="1">Wheel assembly, rear</ci>
</compl>
</compl>
</partasm>
```

Creating the heading for a component list

Use the PartAsm (part assembly) tag to begin a component list. You need to enter a title for the assembly. You can get the heading of the bicycle example by entering these lines of markup:

```
<partasm id="bike" style="bkm:(layout=same)">
<title>Bicycle</title>
```

The formatter provides the following parts of the heading: the prefix, "Assembly" the number of the assembly, beginning with 1 and continuing with increments of 1 in succeeding assembly numbers the colon following the assembly number. "Bicycle" is the name we chose for our assembly. There are other things you can do with the PartAsm tag to make your component list easier to use; we used the BookMaster LAYOUT of SAME to tell BookMaster to place the artwork and as much of the component list as will fit on the same page.

A word about the artwork: Use the MMObj tag to include the drawing of your assembly. We use it after the Title tag. Here's what it looked like in our bicycle assembly markup:

```
<mmobj><objref obj="bike">
<textalt>Bicycle</textalt>
</mmobj>
```

For more information about artwork, see "Including artwork in documents" on page 55.

Developing the component list

Now that we've discussed the beginning of a parts list, let's take a look at the markup we used to create the component list for Assembly 1.

```
<compl>
<ci idxnum="1" partnum="4563423" upa="1">Bike</ci>
<compl>
<ci idxnum="1" partnum="1230987" upa="1">Frame</ci>
<ci idxnum="2" partnum="1238475" upa="1">Wheel assembly, front</ci>
<ci idxnum="3" partnum="1234939" upa="1">Wheel assembly, rear</ci>
</compl>
</compl>
```

Use the Compl (component list) tag to begin each component list. It has no attributes and no text, so it looks quite simple. When the formatter encounters the first Compl tag after a PartAsm tag, it supplies the column headings and the lines that make up the framework that encloses the catalog list and separates the columns.

Often an item in a component list is made up of other items. In order to show this hierarchy, you must nest them in your markup. That is, you must put component lists within other component lists. Component lists may be nested up to three deep. Remember, each component list must begin with its own Compl tag.

Use one CI (component item) tag for each item you want in your list. Whenever we use parts catalogs, we expect to find certain standard information, like a catalog number for ordering parts, a callout number so we can locate the part on the assembly drawing, and a number telling us how many of these items are in the assembly. We use attributes on the CI tag to add that information to our com-ponent lists, so the CI tag with all its attributes looks a lot more complicated than it really is. Here's what one of the CI tags from our example looks like:

```
<ci idxnum="1" partnum="4563423" upa="1">Bike</ci>
```

Now let's look at each of the attributes and how to use them:

ID Use the ID attribute when you need to identify a component item so that cross references can be made to it.

IDXNUM
Use the IDXNUM attribute to assign to the item a number that matches an

artwork index (callout) number. The number you assign with the IDXNUM attribute shows up in the “Asm-Index” column and is prefixed with a dash character. The number to the left of the dash is the same assembly number that the formatterused in the heading prefix.

PARTNUM
Use the PARTNUM attribute to assign the item’s part number. The number you assign with the PARTNUM attribute shows up in the “Part Number” column. Part numbers are limited to seven alphanumeric characters (A–Z, a–z, 0–9), with no intervening blanks.

UPA
Use the UPA (units per assembly) attribute to tell how many of this particular item there are in the assembly. The number you assign with the UPA attribute shows up in the “Units” column.

Including comments in the component list

You can use the CompCmt (component comment tag) to include helpful information that is not part of a component description. Just enter the text of your comment inside CompCmt tag. The comment text you enter appears indented in the “Description” column. Here’s what the CompCmt tag line from our front wheel example on page 216 looks like.

```
<compcmt>For detailed breakdown, see <xref refid="wheelxmp">.
</compcmt>
```

Cross-referencing part assemblies and component lists

Often we need to tell the readers of our component list where to find other related component lists or component items. Usually we want to point them to another assembly that shows a more detailed breakdown of a particular item. Sometimes we want to point them to another component that shows where a particular item fits in a larger assembly. In both cases, we must first use the ID attribute to identify.the target (what we’re referring to), then use either the CIREF or the XREF tag to point to the target. Because we need another component list to show you how to refer from one component list to another, here’s the markup for a second assembly and component list.

```
<partasm id="wheelxmp" style="bkm:(layout=same)">
<title>Wheel, front</title><compl>
<ci partnum="56-2345">Wheel assembly, front</ci>
<compcmt>For next higher assembly see <xref refid="bike">.</compcmt>
<ci idxnum="1" partnum="33-5234" upa="1">Tire, clincher 27 x 1.125</ci>
<ci idxnum="2" partnum="56-4352" upa="1">Tube, 27 x 1.125</ci>
<ci idxnum="3" partnum="56-3489" upa="1">Rim liner</ci>
<ci idxnum="4" partnum="56-6534" upa="1" id="wheel2a">Wheel assembly</ci>
<compl>
<ci idxnum="5" partnum="56-3476" upa="1">Rim, aluminum alloy 27 x 1.125</ci>
<ci idxnum="6" partnum="56-8393" upa="36">Spoke, 298mm</ci>
<ci idxnum="7" partnum="56-9845" upa="36">Spoke bolt</ci>
</compl>
<ci idxnum="8" partnum="56-9874" upa="1">Hub assembly, front</ci>
</compl></partasm>
```

Assembly 2: Wheel, front

Asm– Index	Part Number	Units	Description
2–	56-2345		Wheel assembly, front For next higher assembly see “Assembly 1: Bicycle” on page 216.
–1	33-5234	1	Tire, clincher 27 x 1.125
–2	56-4352	1	Tube, 27 x 1.125
–3	56-3489	1	Rim liner
–4	56-6534	1	Wheel assembly
–5	56-3476	1	• Rim, aluminum alloy 27 x 1.125
–6	56-8393	36	• Spoke, 298mm
–7	56-9845	36	• Spoke bolt
–8	56-9874	1	Hub assembly, front

Keeping track of assemblies and parts

The AsmList (assembly list) tag and the PNIndex (part number index) tag help you find each assembly and part.

Getting an assembly list

The AsmList tag works much like a partial table of contents; it gives you an alphabetical listing of the headings from your PartAsm tags, along with the page numbers on which their headings appear.

If you want to put your assembly list in the front matter of your document, here is how you might enter your markup:

```
<frontm style="display='tipage cover'">
<toc><gendtitle></toc>
<d>
<dprolog><titleblk>
<title>List of assemblies</title>
</titleblk></dprolog>
<dbody>
<asmlist>
</dbody></d>
</frontm>
```

You can put your assembly list at the beginning of a chapter instead of in the front matter, like so:

```
<d>
<dprolog><titleblk>
<title>Parts catalog</title>
</titleblk></dprolog>
<dbody>
<p>This portion contains the parts and assembly instructions
for your Mark-21 Super Bi-Pedal Tricycle.</p>
<d>
<dprolog><titleblk>
<title>List of assemblies</title>
</titleblk></dprolog>
<dbody>
<asmlist>
</dbody></d>
<partasm>
  ...
</partasm></dbody></d>
```

But wherever you put it, remember, only one AsmList tag is allowed per document.

Note: ASMLIST is not supported in the HTML output transform.

Getting a part number index

An index of part numbers can help in retrieving individual parts in your document. If you enter the PNIndex tag in the back matter of a document, the formatter sorts the part numbers you entered with the CI tag's PartNum attribute and prints them and their page numbers.

Part numbers from sample parts assemblies are excluded from the part number index. The part number sort sequence is different than that of the regular index; all one-digit part numbers are listed, followed by all two-digit part numbers, and so on. Here's the markup used to get the part number index in this book.

```
<pnindex id="partnumindex">
<gendtitle>
</pnindex>
```

Part 3. IBMIDDoc Markup Reference

Chapter 24. Reference Explanation

This chapter lists the type of information that is provided for each element or attribute in Chapter 25, "IBMIDDoc Elements" on page 231 and describes how to read the syntax diagrams.

Element and Attribute Descriptions

The elements and attributes are listed in alphabetical order. For each element or attribute, the following information is provided:

Name The name and a short description of the element or attribute.

Purpose
The purpose of the element or attribute.

Examples
One or more examples showing how the element or attribute is used.

Attributes and contained elements
Descriptions of attributes, contained elements, and attribute values.

Usage Description of how to use the element or attribute.

Contexts
A list of the elements that can directly contain the element or have the attribute, or a description of where the element or attribute can be used.

How to Read the Syntax Diagrams

This section describes how to read and use the syntax diagrams, which define the rules for typing element markup in a text-editing environment such as XEDIT or EPM. For more information about markup, see "Markup Rules" on page 11.

- Read the diagrams from left-to-right, top-to-bottom, following the main path line. Each diagram begins on the left with double arrowheads (>>) and ends on the right with two arrowheads facing each other (><).
- If a diagram is longer than one line, each line to be continued ends with a single arrowhead (>) and the next line begins with a single arrowhead.
- A word that is not in italics is an operand or value you must spell exactly as shown. However, you can enter it using any case.

```
►►──OPERAND──────────────────────────────────────────────────────────►◄
```

  If an operand or value can be abbreviated, the abbreviation is discussed in the text associated with the syntax diagram.
- A word in italics is a *variable*. Where you see a variable in the syntax, you must replace it with one of its allowable names or values, as defined in the text.

```
►►──variable─────────────────────────────────────────────────────────►◄
```

- Single-word attribute values are not shown with quotation marks (but any attribute value can be entered with quotation marks around it). Multiple-word attribute values and any attribute value that contains special characters must be enclosed in quotation marks. Quotation marks are always shown as double quotation marks ("), but single quotation marks (') can be used unless the value contains single quotation marks or an apostrophe. For more information about markup with quotation marks, see “Markup Rules” on page 11.

- Required operands and values appear on the main path line. You must code required operands and values.

```
►►──REQUIRED_OPERAND──────────────────────────────────────────────────►◄
```

  If several mutually exclusive required operands or values exist, they are stacked vertically in alphanumeric order.

```
►►──┬─REQUIRED_OPERAND_OR_VALUE_1─┬───────────────────────────────────►◄
    └─REQUIRED_OPERAND_OR_VALUE_2─┘
```

- Optional operands and values appear below the main path line. You can choose not to code optional operands and values.

```
►►──┬─────────┬───────────────────────────────────────────────────────►◄
    └─OPERAND─┘
```

  If several mutually exclusive optional operands or values exist, they are stacked vertically in alphanumeric order below the main path line.

```
►►──┬───────────────────┬─────────────────────────────────────────────►◄
    ├─OPERAND_OR_VALUE_1─┤
    └─OPERAND_OR_VALUE_2─┘
```

- Default operands and values appear above the main path line. If you omit the operand entirely, the default is used.

```
    ┌─DEFAULT─┐
►►──┼─────────┼───────────────────────────────────────────────────────►◄
    └─OPERAND─┘
```

- An arrow returning to the left above an operand or value on the main path line means that the operand or value can be repeated. The comma means that each operand or value must be separated from the next by a comma. If a space is shown, each operand or value must be separated from the next by a space.

```
    ┌─,────────────────┐
    ▼                  │
►►────REPEATABLE_OPERAND─┴────────────────────────────────────────────►◄
```

- An arrow returning to the left above a group of operands or values means that more than one can be selected or that a single one can be repeated.

```
►►─┬─────────────────────────────────────┬──────────────────────────►◄
   │   ┌─,───────────────────────────┐   │
   │   ▼                             │   │
   └───┬─REPEATABLE_OPERAND_OR_VALUE_1─┬─┴─┘
       └─REPEATABLE_OPERAND_OR_VALUE_2─┘
```

- References to syntax notes appear as numbers enclosed in parentheses above the line. Do not code the parentheses or the number.

```
►►──OPERAND─────────────────────────────────────────────────────────►◄
```

Notes:

1 An example of a syntax note.

- Some diagrams contain *syntax fragments*, which serve to break up diagrams that are too long, too complex, or too repetitious. Syntax fragment names are in mixed case and are shown in the diagram. The fragment is placed either below the main diagram or in a separate description.

```
►►──┤ Syntax Fragment ├─────────────────────────────────────────────►◄
```

Syntax Fragment:

```
├──1ST_OPERAND,2ND_OPERAND,3RD_OPERAND──────────────────────────────┤
```

Common Element Attributes (large set)

Several elements are defined to use this set of attributes:

Class
The Class attribute associates an element class with an element. This attribute must contain an SGML name that has been defined as a class name in a ClassDef element. This attribute only applies to elements specified on the ClassDef's ELETYPE attribute. Element classes are defined with ClassDef elements within PropDefs. See Chapter 20, "Property and Class Definitions" on page 201 for more information.

Conloc
The CONLOC attribute specifies that the content of another element of the same type is to be used as the content of the referencing element. This enables reuse of information. When the CONLOC attribute is specified, you cannot specify the element's end tag. The result of using CONLOC is exactly the same as if the element being referred to had occurred at that point in the document. See "Reusing elements from an object library" on page 191.

Attributes on the element are now passed to the element with the CONLOC; this is a feature that began with IDWB release 3.4, patch IDWXF036.

HyTime
ignored by processes

ID=*identifier*
The ID attribute identifies an element within an SGML document. IDs must be

unique within a single document. Any element that has an ID can be cross-referenced or linked to. IDs can be up to 64 characters long. IDs must start with an alphabetic character and can contain letters, numbers, dashes (-), or periods (.).

InfoMast
A fixed attribute used to classify the element.

Props=*properties*
The Props attribute is used to specify the condition under which the information contained within the element appears. See "Property-Based Retrieval" on page 195.

PropSrc
Points to an element whose properties are to be used as the properties of the referencing element. See Chapter 20, "Property and Class Definitions" on page 201.

Qualif
The QUALIF attribute refers to the ID of a qualification element. See "Qualifying information" on page 49.

Reftype
ignored by processes

RetAlts
The RETALTS attribute points to one or more elements whose content may be used in place of, or in addition to, the referencing element's content. This attribute must reference one or more elements of the same element type. This attribute usually references elements in an object library. See "Retrieval alternatives" on page 198.

Rev
The REV attribute references the Rev element which describes the last revision level of the information. See "Using Revisions" on page 109.

Status
ignored by processes

Style
The Style attribute contains either a reference to a separate style specification for an element or a set of element-specific presentation style definitions when the processing system does not support separate styles for elements. Assigning a value to the STYLE attribute modifies how an element is presented when it is output.

XrefText
The XRefText attribute defines the text to be used when an element is the target of a link that generates a reference. See Chapter 6, "Cross-referencing" on page 61.

Common Element Attributes (small set)

Several elements are defined to use this set of attributes:

Class
The Class attribute associates an element class with an element. This attribute must contain an SGML name that has been defined as a class name in a ClassDef element. This attribute only applies to elements specified on the

ClassDef’s ELETYPE attribute. Element classes are defined with ClassDef elements within PropDefs. See Chapter 20, “Property and Class Definitions” on page 201 for more information.

ID=*identifier*
The ID attribute identifies an element within an SGML document. IDs must be unique within a single document. Any element that has an ID can be cross-referenced or linked to. IDs can be up to 64 characters long. IDs must start with an alphabetic character and can contain letters, numbers, dashes (-), or periods (.).

Props=*properties*
The Props attribute is used to specify the condition under which the information contained within the element appears. See “Property-Based Retrieval” on page 195.

PropSrc
Points to an element whose properties are to be used as the properties of the referencing element. See Chapter 20, “Property and Class Definitions” on page 201.

Rev
The REV attribute references the Rev element which describes the last revision level of the information. See “Using Revisions” on page 109.

Status
ignored by processes

Style
The Style attribute contains either a reference to a separate style specification for an element or a set of element-specific presentation style definitions when the processing system does not support separate styles for elements. Assigning a value to the STYLE attribute modifies how an element is presented when it is output.

HyTime
ignored by processes

InfoMast
A fixed attribute used to classify the element.

Reftype
ignored by processes

Chapter 25. IBMIDDoc Elements

This section describes the elements and attributes in the IBMIDDoc language.

Abbrev (abbreviations)

Purpose

The Abbrev element is a special division, and contains an explanation of abbreviations used in the document. The best way to create a list of abbreviations is to use the DL element.

Examples

```
<abbrev>
<specdprolog><gendtitle></specdprolog>
<dbody>
<dl>
<dlentry><term>IBMIDDoc</term>
<defn>IBMIDDoc is the name of IBM's implementation
of the SGML standard for software documentation.</defn>
</dlentry>
</dl>
</dbody></abbrev>
```

Attributes

Toc=toc | notoc
Specifies whether the heading should appear in the table of contents. The default is for headings to appear in the table of contents; to a certain level (such as heading level 3). This replaces the older style=hidden attribute. Use this to prevent headings from appearing in the table of contents. It cannot be used to add a heading to the table of contents that would not normally appear. The levels of headings that appear in the table of contents are determined by the MaxToc attribute of the IBMIDDoc tag.

| **LAYOUT=Default-Layout | OneCol | OffsetCol | TwoCol**
| Specifies the column-style for the book.

| **Default-Layout**
| The section uses the default layout for the document style.

| **OneCol**
| The headings and text format across the entire page.

| **OffsetCol**
| Formats the text using an indented, one-column layout. The headings
| format across the page, with the offset text, or indented from the
| margin.

| **TwoCol**
| The text formats in two columns. Headings format across the page or
| with the two-column text.

See “Common Element Attributes (large set)” on page 227.

Abbrev (abbreviations)

Usage

See “Special sections” on page 100.

Contexts

Children: DBody, DIntro, DSum, SpecDProlog.

Parents: BackM, FrontM.

Abstract (abstract)

Purpose

The Abstract special division element contains a short description of the content of the document. Use Abstract to contain a brief description of the document.

Examples

```
<abstract>
<specdprolog><gendtitle></specdprolog>
<dbody>
<p>This describes how to...</p>
</dbody></abstract>
```

Attributes

| **LAYOUT=Document-Layout | OneCol | TwoCol | OffsetCol**
| Specifies the column-style for this portion of the book.

| **Document-Layout**
| The section uses the default layout for the document style.

| **OffsetCol**
| Formats the text using an indented, one-column layout. The headings
| format across the page, with the offset text, or indented from the
| margin.

| **OneCol**
| The text formats across the entire page.

| **TwoCol**
| The text formats in two columns.

Toc=toc | notoc
Specifies whether the heading should appear in the table of contents. The default is for headings to appear in the table of contents; to a certain level (such as heading level 3). This replaces the older style=hidden attribute. Use this to prevent headings from appearing in the table of contents. It cannot be used to add a heading to the table of contents that would not normally appear. The levels of headings that appear in the table of contents are determined by the MaxToc attribute of the IBMIDDoc tag.

See “Common Element Attributes (large set)” on page 227.

Usage

See “Special sections” on page 100.

Contexts

Children: DBody, DIntro, DSum, SpecDProlog.

Parents: D, FrontM, Part.

Address (address)

Purpose

The Address element contains the address of a person or corporation. Address is normally used within the context of an Author element but may be used elsewhere. Enter the address text in the form you want it to be displayed. The text will not be reflowed when the markup is processed. Contained elements will be processed according to the styles used by the processing application.

Examples

```
<authors>
<author><person>
<name>Fred Mertz</name>
<address>
125 West Hollywood Blvd
Tinseltown, CA <postalcode>90210</postalcode></address>
</person></author>
</authors>
```

Attributes

OBJ=*file-entity-name*
The OBJ attribute names a file entity that contains the text for the address.

See “Common Element Attributes (large set)” on page 227.

Usage

See “Author and Address” on page 89.

Contexts

Children: text (#pcdata), IBMMail, Internet, L, Ph, Phone, PostalCode, Term, TM, VNET, WebPage.

Parents: AnnotBody, Attention, Bridge, Caution, CompCmt, Corp, Danger, Defn, Desc, entry, Fn, L, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, NoteBody, P, Person, Ph, Publisher, Q, SynNote, Warning.

Annot (annotation)

Purpose

Use Annot to annotate the content of its containing element, such as notes to reviewers or editors. Annotations can be presented or suppressed, depending on the options given to the processing system.

Migration Notes

- Annot cannot be used to comment out information.
- Unlike the BookMaster Annot element, Annot cannot span document structures. It is an element in the document hierarchy, like any other element.

Examples

```
<P>This text is a paragraph before the annotation.</p>
<ANNOT>
<ANNOTBODY>
<P>This is an annotation, the first paragraph.
<P>This is the second paragraph of the annotation
</ANNOTBODY>
</ANNOT>
<P>This is a paragraph after the annotation.</p>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Annotations” on page 49.

Contexts

Children: AnnotBody, Title.

Parents: Attention, Bridge, Caution, Danger, DBody, Defn, DIntro, DSum, entry, Fig, FigSeg, Fn, LEDI, LI, LQ, MkNote, ModDesc, ModItem, MsgItem, NoteBody, P, PBlk, ProcIntro, SynNote, Warning.

AnnotBody (annotation body)

Purpose

Use AnnotBody to contain the body of the annotation; see “Annot (annotation)” on page 233.

Examples

```
<P>This text is a paragraph before the annotation.</p>
<ANNOT>
<ANNOTBODY>
<P>This is an annotation, the first paragraph.
<P>This is the second paragraph of the annotation
</ANNOTBODY>
</ANNOT>
<P>This is a paragraph after the annotation.</p>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Annotations” on page 49.

Contexts

Children: text (#pcdata), Address, APL, Attention, Bin, Bridge, Caution, CGraphic, Char, Cit, Danger, Date, Dec, DL, Fig, Formula, GL, Hex, L, Lines, LitData, LQ, MD, MkNote, MMObj, ModInfo, MV, Note, NoteList, Num, Oct, OL, P, ParmL, PBlk, Ph, PK, PV, Q, RefKey, Screen, SynPh, Syntax, Table, Term, TM, UL, Xmp, XPh, XRef.

Parents: Annot.

APL (APL data)

Purpose

Use the APL element to identify data that is part of an APL program. The content of the element may be APL data or other elements that make sense in the APL context. This data is encoded in the document source using the character encoding used for APL data and programs, not necessarily the character encoding used for the data everywhere else in the document. The content of this element is typically presented using the same font as is conventionally used for APL, which will also probably differ from that used to present the other data found in the document. An external entity containing the APL data may be referred to using the OBJ attribute, which must contain the name of a data entity. Character entities can also be used to represent APL characters.

Examples

```
<p>A matrix is defined with the string:
<APL>2 3ρ1 2 3 4 5 6</APL></p>
```

Attributes

OBJ=*file-entity-name*
The name of the file entity that contains the APL data.

Notation=apl
Specifies that the notation of the content is in APL format.

See “Common Element Attributes (large set)” on page 227.

Usage

See “Highlighting” on page 43.

Contexts

Children: text (#pcdata).

Parents: AnnotBody, Attention, Bridge, Caution, CompCmt, Danger, Defn, Desc, entry, Fn, L, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, MsgText, NoteBody, P, Ph, Q, SynNote, Term, Warning.

Appendix

Purpose

The Appendix element contains division-like elements that are to be considered appendixes. Appendixes are not considered part of the content of the main body of the SGML markup. They usually contain reference information. In the default presentation style, appendixes are numbered with letters rather than digits.

Examples

```
<backm>
<appendix>
<d>
<dprolog><titleblk>
<title>Special stuff</title>
</titleblk></dprolog>
<dbody></dbody></d>
</appendix></backm>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Using appendix” on page 101.

Contexts

Children: D, DBlk, LERS, ModInfo, MsgList, PartAsm, Proc, RetKey.

Parents: BackM.

Approvers (document approvers)

Purpose

Approvers contains the elements that identify the person or organization who must approve a document or division for publication.

Examples

```
<approvers>
<person><name>Ethel Mertz</name></person>
</approvers>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Contexts

Children: Corp, Person.

Parents: DProlog, Prolog, SpecDProlog.

AreaDef (defines graphic hot spot area)

Purpose

This is currently not working in the ID Workbench transforms.

The AreaDef element contains the specifications of a graphic hot spot. The geometry of graphic hot spots is specified according to to the shape of the hot spot. The numbers specified represent pels in the bitmap for bitmaps, and represent quanta in the underlying grid used in specifying points in a vector graphic. Multiple AreaDef elements can be used in a single MMObjLink element to indicate that more than one area in the graphic can be used to invoke the link.

Examples

```
<mmobj><objref obj="bear"><mmobjlink linkend="a">
<areadef shape="circle" coords="10 15 20"></areadef>
</mmobjlink>
<textalt>One teddy bear.</textalt>
</mmobj>
```

Attributes

SHAPE = rectangle|circle|polygon
describes the shape of the graphic hot spot.

COORDS = *numbers*
contains the coordinates for the graphic hot spot. Values are blank delimited.

rectangle
left *x* axis, top *y* axis, right *x* axis, bottom *y* axis.

circle
center *x*, center *y*, radius, with center specified relative to the origin of the graphic.

polygon
1st *x*, 1st *y*, 2nd *x*, 2nd *y*, Nth *x*, Nth *y* with automatic closure if the first and last point are not identical. It must be an error if a line drawn between any two adjacent points intersects with any other line drawn between any other two adjacent points in the specification.

See "Common Element Attributes (large set)" on page 227.

Contexts

Children: TextAlt.

Parents: LDescs, MMObjLink.

AsmList (list of parts assemblies)

Purpose

The AsmList element is a specialized list element that contains a list of all parts assembly lists in the document.

Examples

```
<ASMLIST>
```

Attributes

SPEC=AUTO|MAN
This attribute has a fixed value of AUTO, generate the list from the assemblies in the document.

See "Common Element Attributes (large set)" on page 227.

AsmList (list of parts assemblies)

Usage

See “Getting an assembly list” on page 219.

Contexts

Children: empty.

Parents: DBody, DIntro, DSum, LEDI, MsgItem, PBlk, ProcIntro.

Attention (safety notice)

Purpose

Use an Attention notice to indicate the possibility of damage to a program, device, system, or data.

Examples

```
<attention>Here's a way to get someone's attention.
</attention>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “The perils of processing: Attention, caution, and danger” on page 48.

Contexts

Children: text (#pcdata), Address, Annot, APL, Bin, Bridge, CGraphic, Char, Cit, Date, Dec, DL, Fig, Formula, GL, Hex, L, Lines, LitData, LQ, MD, MkNote, MMObj, ModInfo, MV, Num, Oct, OL, P, ParmL, PBlk, Ph, PK, PV, Q, RefKey, Screen, SynPh, Syntax, Table, Term, TM, UL, Xmp, XPh, XRef.

Parents: AnnotBody, Bridge, DBody, Defn, DIntro, DSum, entry, LEDI, LI, LQ, ModDesc, ModItem, MsgItem, PBlk, ProcEntry, ProcIntro, Safety.

Author

Purpose

Use Author to contain information about an author, such as name, title, and so forth.

Examples

```
<authors>
<author><person>
<name>Fred Mertz</name>
<address>125 West Hollywood Blvd
Tinseltown, CA <postalcode>90210</postalcode></address>
</person></author>
</authors>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Author and Address” on page 89.

Contexts

Children: Corp, Person, Title.

Parents: Authors, Rev.

Authors

Purpose

The Authors element contains information about one or more authors of the document.

Examples

```
<authors>
<author><person>
<name>Fred Mertz</name>
<address>125 West Hollywood Blvd
Tinseltown, CA <postalcode>90210</postalcode></address>
</person></author>
</authors>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Author and Address” on page 89.

Contexts

Children: Author, Desc.

Parents: BibEntry, DProlog, IBMBibEntry, SpecDProlog.

BackCover (back cover)

Purpose

The BackCover element contains a reference to the art used for the document’s back cover.

Examples

```
<ibmbibentry><doctitle><titleblk>
<title>My Document</title>
</titleblk></doctitle>
<ibmdocnum></ibmdocnum>
<coverdef><frontcover><mmobj><objref obj="front1">
<textalt></textalt></mmobj></frontcover>
<backcover><mmobj><objref obj="back1">
<textalt></textalt></mmobj></backcover>
</coverdef></ibmbibentry>
```

BackCover (back cover)

Usage

See "Adding to the front or back cover (CoverDef)" on page 91.

Contexts

Children: BibList, CGraphic, DL, Lines, LitData, MMObj, OL, P, Table, UL, Xmp.

Parents: CoverDef.

BackM (back matter)

Purpose

The BackM element contains the material that follows the body of a document. It may include appendixes, a glossary, and an index. The output transforms automatically provide a part separator for the back matter when the body of the document contained a Part tag. If you want to suppress this part separator, use the following coding on the Backm tag:

```
<backm style="xpp:(nopart)">
```

Examples

```
<BACKM>
 <APPENDIX>
  <D>
   <DPROLOG>
    <TITLEBLK>
     <TITLE>WHANTOOZLER TECHNICAL SPECIFICATIONS
     </TITLE>
    </TITLEBLK>
   </DPROLOG>
   <DBODY>
    <P>THIS SECTION DESCRIBES THE ELECTRICAL CONFIGURATION
       OF THE.....
    </P>
   </DBODY>
  </D>
 </APPENDIX>
</BACKM>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See "About back matter (BackM)" on page 101.

Contexts

Children: Abbrev, Appendix, Bibliog, D, DBlk, Glossary, Index, MasterIndex, PNIndex, RCF, SOA.

Parents: IBMIDDoc.

BibEntry (bibliographic entry)

Purpose

The BibEntry element contains information about a document. The IBMBibEntry element is used to define bibliographic entries for IBM documents. You can use this to create bibliographic information for non-IBM bibliographies and title citations.

Examples

To create a bibliography definition:

```
<bibentrydefs><bibentry docname="dislike"><doctitle>
<titleblk><title>Things I Dislike</title></titleblk>
</doctitle></bibentry></bibentrydefs>
```

Attributes

DOCLINK=*ID*
: The DocLink attribute specifies the ID of the URL defined on a Notloc element.

DOCNAME=*entity_name*
: Contains a reference to the ID or name of an entity that is defined in the document that must also be referenced by a NameLoc element. This indicates a cross-document target with the specified ID value.

See "Common Element Attributes (large set)" on page 227.

Usage

See Chapter 14, "Bibliographies and citations" on page 141.

Contexts

Children: Authors, Desc, DocTitle, ExternalFileName, ISBN, OrderNum, PrtLoc, PublicId, Publisher.

Parents: BibEntryDefs, BibList, Cit.

BibEntryDefs (contains bibliographic entries)

Purpose

Use the BibEntryDefs element to contain BibEntry and LibEntry elements. When used in a DProlog element, BibEntryDefs contains elements used within that division. When used in the Prolog element, BibEntryDefs contains elements used in the document.

Examples

To create a bibliography definition:

```
<bibentrydefs><bibentry docname="dislike"><doctitle>
<titleblk><title>Things I Dislike</title></titleblk>
</doctitle></bibentry></bibentrydefs>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

BibEntryDefs (contains bibliographic entries)

Usage

See Chapter 14, “Bibliographies and citations” on page 141.

Contexts

Children: BibEntry, IBMBibEntry, IBMLibEntry, LibEntry.

Parents: DProlog, Prolog, SpecDProlog.

Bibliog (bibliography)

Purpose

The Bibliog special division contains lists of documents and other materials relevant or related to a document. You can use BibList elements within Bibliog to contain or generate bibliography lists.

Examples

```
<bibliog>
<specdprolog><gendtitle></specdprolog>
<dbody>
<biblist><bibentry><doctitle><titleblk><title>My Nice
Book</title></titleblk></doctitle></bibentry>
<bibentry><doctitle><titleblk><title>Your Nice Book
</title></titleblk></doctitle></bibentry>
</biblist>
</dbody></bibliog>
```

Attributes

| **LAYOUT=Document-Layout | OneCol | TwoCol | OffsetCol**
| Specifies the column-style for this portion of the book.

| **Document-Layout**
| The section uses the default layout for the document style.

| **OffsetCol**
| Formats the text using an indented, one-column layout. The headings
| format across the page, with the offset text, or indented from the
| margin.

| **OneCol**
| The text formats across the entire page.

| **TwoCol**
| The text formats in two columns.

Toc=toc | notoc
Specifies whether the heading should appear in the table of contents. The default is for headings to appear in the table of contents; to a certain level (such as heading level 3). This replaces the older style=hidden attribute. Use this to prevent headings from appearing in the table of contents. It cannot be used to add a heading to the table of contents that would not normally appear. The levels of headings that appear in the table of contents are determined by the MaxToc attribute of the IBMIDDoc tag.

See “Common Element Attributes (large set)” on page 227.

Usage

See "Using bibiography (Bibliog)" on page 102.

Contexts

Children: DBody, DIntro, DSum, SpecDProlog.

Parents: BackM, FrontM.

BibList (bibliography entry list)

Purpose

The BibList element either generates or contains a list of bibliography entries.

You can create an explicit bibliography list by building it out of any combination of bibliography entries, library entries, or Cit elements.

Examples

```
<bibliog>
<specdprolog><gendtitle></specdprolog>
<dbody>
<biblist><bibentry><doctitle><titleblk><title>My Nice
Book</title></titleblk></doctitle></bibentry>
<bibentry><doctitle><titleblk><title>Your Nice Book
</title></titleblk></doctitle></bibentry>
</biblist>
</dbody></bibliog>
```

Attributes

SPEC= AUTO
Specifies that the content of the element is generated from bibliographic references from the body of the document.

ENTRYTYPE=DOC | LIB | DOCORLIB
Indicates whether or not the generated list is to contain BibEntry entries or LibEntry entries. If DOCORLIB is used, a list that includes both can be generated. DOC is the default value.

FORM= NORMAL | FULL | TITLE | DOCNUM
Defines the form of the generated bibliography entries. The specific meaning of **FULL** and **NORMAL** is defined by the active style. NORMAL is the default value.

See "Common Element Attributes (large set)" on page 227.

Usage

See Chapter 14, "Bibliographies and citations" on page 141.

Contexts

Children: BibEntry, Cit, IBMBibEntry, IBMLibEntry, LibEntry.

Parents: BackCover, DBody, DIntro, DSum, FrontCover, LEDI, MsgItem, PBlk, ProcIntro.

Bin (binary data)

Purpose

The Bin element contains text representing binary data.

Examples

```
<BIN>11000001</BIN>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Highlighting” on page 43.

Contexts

Children: text (#pcdata).

Parents: AnnotBody, Attention, Bridge, Caution, CompCmt, Danger, Defn, Desc, entry, Fn, L, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, MsgText, NoteBody, P, Ph, Q, SynNote, Term, Warning.

Body (document body)

Purpose

Use Body to contain the main body of your document.

Examples

```
<body>
<d>
<dprolog><titleblk>
<title>Introduction</title>
</titleblk></dprolog>
<dbody>
<p>Please type your text here. Thank-you.</p>
</dbody></d>
<d>
<dprolog><titleblk>
<title>Caring for your fruit bat</title>
</titleblk></dprolog>
<dbody>
<p>This contains all sorts of information</p>
</dbody></d>
</body>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Creating the body of your document” on page 20.

Contexts

Children: D, DBlk, LERS, ModInfo, MsgList, Part, PartAsm, Proc.

Parents: IBMIDDoc.

BOFNum (bill of forms number)

Purpose

The BOFNum element contains the bill of forms number assigned to the library described by the LibEntry element.

Examples

```
<libentry>
<library><titleblk><title>My Title</title></titleblk>
</library>
<bofnum>SBOF-1234</bofnum>
</libentry>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Contexts

Children: text (#pcdata), Ph.

Parents: LibEntry.

Bridge (bridge between concepts)

Purpose

Use the Bridge element to link two locations in a document together and to explain the linkage. The simplest use of Bridge is to create a bridging title between items in a list. A good example of bridges are the titles used throughout magazine articles to bridge a reader from one topic to the next. Such titles do not define hierarchical divisions, but serve merely as a transition from one part to the next.

Examples

This example shows bridging of two sets of list items:

```
<ol>
<li>Saute the shallots and chopped mushrooms until
the shallots are tender and the liquid from the mushrooms
has cooked away.</li>
<li>Brown the sausage and add to the mushroom mixture.
</li>
<bridge><p>The above may be prepared several hours in
advance and refrigerated. Then, 30 minutes before
serving time, finish the dish</p></bridge>
<li>Mix one can of tomato sauce with the mushroom
and sausage mixture and bring to a slow simmer.
</li>
<li>Add the heavy cream and immediately pour into
a casserole.</li>
<li>Pop into 350-degree oven for 15 minutes.</li>
</ol>
```

Bridge (bridge between concepts)

Attributes

LINKENDS=*element_id1 element_id2*
The *element_id*s are optional identifiers of the element locations that are to be linked or bridged. If the IDs are not specified, the element to be linked is defaulted. For *element_id1*, the default is the element preceding the Bridge element. For *element_id2*, the default is the element following the Bridge element. If you specify only one of the link ends explicitly, you must specify the keyword **#IMPLIED** for the other element.

See "Common Element Attributes (large set)" on page 227.

Usage

See "Separating or bridging list items" on page 38.

Contexts

Children: text (#pcdata), Address, Annot, APL, Attention, Bin, Caution, CGraphic, Char, Cit, Danger, Date, Dec, DL, Fig, Formula, GL, Hex, L, Lines, LitData, LQ, MD, MkNote, MMObj, ModInfo, MV, Note, NoteList, Num, Oct, OL, P, ParmL, PBlk, Ph, PK, PV, Q, RefKey, Screen, SynPh, Syntax, Table, Term, Title, TM, UL, Xmp, XPh, XRef.

Parents: AnnotBody, Attention, Caution, Danger, DBody, Defn, DIntro, DL, DLBlk, DSum, entry, Fig, FigSeg, Fn, GL, GLBlk, LEDI, LI, LIBlk, LQ, MkNote, ModDesc, ModItem, MsgItem, NoteBody, NoteList, OL, P, ParmBlk, ParmL, PBlk, ProcIntro, StepNotes, SynNote, UL, Warning.

Cap (caption)

Purpose

The Cap element contains a caption for Figure or Table element. The caption text can appear in figure and table lists. It should be a relatively short description of the figure or table.

Examples

```
<fig id="pubhist">
<cap>History of Publishing within IBM</cap>
<mmobj><objref obj="pubhist">
<textalt>First, there was the pencil, which begat
the pen and the typewriter. Then came ATMS, Script/DCF,
ISIL, BookMaster, and IBMIDDoc.</textalt>
</mmobj></fig>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See "Figure captions and descriptions" on page 57 and "Table captions and descriptions" on page 69.

Contexts

Children: text (#pcdata), L, Ph, Term, TM.

Parents: Fig, Table.

Caution (caution notice)

Purpose

Use Caution to create a caution notice, consisting of one or more paragraphs or other paragraph-level elements. Cautions are normally used to warn about actions that may cause damage to equipment.

Examples

```
<CAUTION>Running your engine without oil may cause irreparable damage.
</CAUTION>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “The perils of processing: Attention, caution, and danger” on page 48.

Contexts

Children: text (#pcdata), Address, Annot, APL, Bin, Bridge, CGraphic, Char, Cit, Date, Dec, DL, Fig, Formula, GL, Hex, L, Lines, LitData, LQ, MD, MkNote, MMObj, ModInfo, MV, Num, Oct, OL, P, ParmL, PBlk, Ph, PK, PV, Q, RefKey, Screen, SynPh, Syntax, Table, Term, TM, UL, Xmp, XPh, XRef.

Parents: AnnotBody, Bridge, DBody, Defn, DIntro, DSum, entry, LEDI, LI, LQ, ModDesc, ModItem, MsgItem, PBlk, ProcEntry, ProcIntro, Safety.

CGraphic (character graphic)

Purpose

The CGraphic element contains a graphic created with box and line characters. See “Entities” on page 5 for information about declaring external data entities.

Migration Note

BookMaster CGraphics can contain characters that are not part of the IBMIDDoc document character set and thus **must** be made into external entities.

Examples

```
<!entity mygraphic SYSTEM "mygraph.cgr" ndata linespec>
 ...
<FIG>
<CAP>Simple box and line graphic</CAP>
<CGRAPHIC OBJ="mygraphic">
</FIG>
```

CGraphic (character graphic)

Attributes

OBJ=*data-entity-name*
: Specifies the SGML file that contains the character graphic. When you specify OBJ, do not include the CGraphic end tag. You must use a notation of LINESPEC for a CGraphic.

| **LINELENGTH=***characters*
| This specifies the number of characters in widest line. If the number of
| characters does not fit in the column or page width, the text size is reduced.

See “Common Element Attributes (large set)” on page 227.

Usage

See “Character graphics” on page 59.

Contexts

Children: text (#pcdata), L, LitData, Ph, RefKey, Term.

Parents: AnnotBody, Attention, BackCover, Bridge, Caution, Danger, DBody, Defn, DIntro, DSum, entry, Fig, FigSeg, Fn, FrontCover, LEDI, LI, LQ, MkNote, ModDesc, ModItem, MsgItem, NoteBody, P, PBlk, ProcIntro, SynNote, Warning.

Char (character data)

Purpose

Use the Char element to identify literal character data.

Examples

```
<P>Enter this character string to indicate
cartoon cussing: <CHAR>%#$@!@#</CHAR>.
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Highlighting” on page 43.

Contexts

Children: text (#pcdata).

Parents: AnnotBody, Attention, Bridge, Caution, CompCmt, Danger, Defn, Desc, entry, Fn, L, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, MsgText, NoteBody, P, Ph, Q, SynNote, Term, Warning.

CI (component item)

Purpose

The CI element contains a component item in an assembly list.

Examples

```
<compl>
<ci idxnum="1" partnum="4563423" upa="1">Bike</ci>
<compl>
<ci idxnum="1" partnum="1230987" upa="1">Frame</ci>
<ci idxnum="2" partnum="1238475" upa="1">Wheel assembly, front</ci>
<ci idxnum="3" partnum="1234939" upa="1">Wheel assembly, rear</ci>
</compl>
</compl>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

IDXNUM
Use the IDXNUM attribute to assign to the item a number that matches an artwork index (callout) number. The number you assign with the IDXNUM attribute shows up in the Asm-Index column and is prefixed with a dash character. The number to the left of the dash is the same assembly number used in the heading prefix.

PARTNUM
Use the PARTNUM attribute to assign the item’s part number. The number you assign with the PARTNUM attribute shows up in the Part Number column. Part numbers are limited to seven alphanumeric characters (A-Z, a-z, 0-9), with no intervening blanks.

UPA
Use the UPA (units per assembly) attribute to tell how many of this particular item there are in the assembly. The number you assign with the UPA attribute shows up in the Units column.

Usage

See Chapter 23, “Creating parts catalog lists” on page 215.

Contexts

Children: text (#pcdata), L, Ph, Term, TM.

Parents: CompL.

Cit (document citation)

Purpose

The Cit element contains a citation to another document.

Examples

Simple citation:

```
<cit><bibentry><doctitle><titleblk><title>Huckleberry
Finn</title></titleblk></doctitle></bibentry></cit>,
by Mark Twain, is a most excellent book.
```

Complex citation:

```
See this book <cit bibid="fruitybat"> and that book
<cit bibid="vampbat" form="full"> for serious bedtime reading.
```

Cit (document citation)

Attributes

BibId=*entry_id*
: Specifies the ID of a bibliographic or library entry defined elsewhere. When BIBID is specified, it is an error to specify any content or the Cit end tag.

FORM=NORMAL | FULL | TITLE | DOCNUM
: Specifies the form of the citation, as follows:

  NORMAL
  : Specifies that the record is to be presented as defined by the active presentation style.

  FULL
  : Specifies that the full range of information in the BibEntry is to be presented.

  TITLE
  : Specifies that only the title is to be presented.

  DOCNUM
  : Specifies that the document number is to be presented.

See “Common Element Attributes (large set)” on page 227.

Usage

See “Simple title citations” on page 45 and “Using title citations” on page 142.

Contexts

Children: BibEntry, IBMBibEntry, IBMLibEntry, LibEntry.

Parents: AnnotBody, Attention, BibList, Bridge, Caution, CompCmt, Danger, Defn, Desc, entry, Fn, L, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, NoteBody, P, Ph, Q, SynNote, Warning.

ClassDef (element class definition)

Purpose

The ClassDef element defines an element class. ClassNames are defined for the document in which the ClassDef element occurs.

Examples

This example shows te common IBM class definitions

```
<classdef classname="ibmcommand" eletypes="ph" style="bold">
<sem>Command names. For example: COPY command</sem>
</classdef>
<classdef classname="ibmemphasis" eletypes="ph" style="italic">
<sem>Text the writer wants to emphasize.</sem>
</classdef>
<classdef classname="ibmfilepath" eletypes="ph">
<sem>File path names. For example: c:\config.sys</sem>
</classdef>
<classdef classname="ibmguicontrol" eletypes="ph" style="bold">
<sem>GUI control names: menu names, menu choices, entry fields,
 icons, folders. list boxes, push buttons,
 radio buttons, spin buttons, or check boxes; NOT:
 windows or notebooks.</sem>
</classdef>
```

Here's a sample use of IBMGuiControl:

```
Press <ph class="IBMGuiControl">OK</ph> to continue."
```

Attributes

See "Common Element Attributes (large set)" on page 227.

CLASSNAME=*classname*
: The name of the class being defined.

ELETYPES=*element names*
: Defines those element types (generic identifiers) to which this ClassDef applies. Use ELETYPES when a ClassDef is only meaningful for a specific set of element types.

STYLE=*styles*
: Specifies the style of the element to assign to the class name.

OutputClass=*css-style-class*
: The outputclass specifies the class style name in the CSS file; see "Using document classes with XHTML style sheets" on page 51.

SEM
: Defines the semantic meaning of a given class. Sem is intended to document what a given element class means to the author that defined it.

Usage

See "Defining Element Classes" on page 202.

Contexts

Children: IdxTerm, Sem, Title.

Parents: PropDefs, PropGroup.

CLE (content list entry)

Purpose

CURRENTLY NOT SUPPORED BY ANY OUTPUT TRANSFORMS.

The CLE tag contains an explicit table of contents, figure list, or table list entry to be used in a content list. The CLE element can also refer to an item to be used in a content list. CLE is used to create entries in content lists manually, when you need a different order or hierarchy than the document. You can use these methods:

- The CLE element can generate a table of contents entry completely by reference. Use this when all of the material is present and the titles are correct; but you want to re-order the entries. This method also allows specific entries to be created or included that are not created during automatic processing.

```
<TOC SPEC="man">
⋮
<CLE REFID="div3">
⋮
</TOC>
⋮
<D ID="div3">Division Three Title
 <DProlog>
```

```
  <TitleBlk>
   <Title>Division Three Title</Title>
  </TitleBlk>
 </DProlog>
```

- You can provide the text for the CLE, but refer to the material that it represents. Use this to specify a different title in the table of contents than division being referenced. In this case, page number is generated from the object reference. The division's STitle could also have been used with a pure reference (the preceeding example).

```
<TOC SPEC="man">
   ⋮
<CLE REFID="div3">Division Three
   ⋮
</TOC>
   ⋮
<D ID="div3">
 <DProlog>
  <TitleBlk>
   <Title>Division Title Number 3</Title>
  </TitleBlk>
 </DProlog>
```

- The CLE may contain the entire entry information, directly in its content. This text may include the leader dots and page numbers. If the CLE is a table of contents entry, the LVL attribute may be used to set the level in the table of contents that the entry should take. If the page number is not included in the CLE content, it may be specified in the STYLE attribute using the Bookmaster page attribute.

```
<TOC SPEC="man">
   ⋮
<CLE LVL="2" STYLE="BKM:(page=134)">entry text shown at level 2
   ⋮
</TOC>
```

Each of the three methods illustrated here may be used in any combination when creating a manual table of contents.

Attributes

See "Common Element Attributes (large set)" on page 227.

REFID=*element_name*
: Contains the ID of the element to which the CLE is referring.

%Title
: Contains the entry text, or one of the %Title elements.

The LVL attribute is used only when the CLE element is used in a TOC, and when the REFID attribute is not used. The LVL attribute contains a number, and is used to specify the level of the TOC entries presented.

Usage

Contexts

Children: text (#pcdata), L, Ph, Term, TM.

Parents: FigList, TList, TOC.

Code (message code number)

Purpose

The Code element contains the number of the code being described by the Msg element.

Examples

```
<MSG><CODE>OOC4</CODE>
<MSGITEM CLASS="xpl">
Occurs when the auto-framatizing circuit blows out.</MSGITEM>
</MSG>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See "Message and code lists" on page 38.

Contexts

Children: text (#pcdata), Ph.

Parents: Msg.

ColSpec (column specification)

Purpose

The ColSpec element contains the specification for a column.

Examples

```
<TABLE FRAME="ALL">
 <TGROUP COLS="4" COLSEP="1" ROWSEP="1" ORIENT="PORT" PGWIDE="0">
  <COLSPEC COLWIDTH="68*">
  <COLSPEC COLWIDTH="127*">
  <COLSPEC COLWIDTH="195*">
  <COLSPEC COLWIDTH="66*">

   ⋮
```

Attributes

See "Common Element Attributes (large set)" on page 227.

COLNUM=*col_number*
: This value indicates the number of the column.

COLNAME=*col_name*
: Specifies the column name. This name can be referenced by other table elements.

ALIGN=LEFT | RIGHT | CENTER | JUSTIFY | CHAR
: This attribute specifies the horizontal positioning of the text contained in the column:

LEFT
: Specifies left alignment (the default).

RIGHT
Specifies right alignment.

CENTER
Specifies center alignment.

JUSTIFY
Specifies that the contained column text is justified,

CHAR
Specifies the character that is used for alignment.

CHAR
specifies the character that is used for alignment.

CHAROFF=*number*
Specifies the character offset for Entry elements in the column.

COLWIDTH=*measure*
Specifies a fixed, proportional, or mixed measure for the column width.

Migration Note
At this time, mixed measures are not supported. You should use proportional measures.

COLSEP=0 | 1
This attribute's value specifies that the internal column rules should be:
- drawn to the right of each cell's content (1)
- not displayed at all (0)

ROWSEP=0 | 1
This attribute's value specifies that the internal row rules should be:
- drawn below each Entry element that ends a row (1)
- not displayed at all (0)

Usage

Chapter 7, "Creating IBMIDDoc Tables" on page 67

Contexts

Children: empty.

Parents: tgroup.

CompCmt (component comment)

Purpose

The CompCmt element contains a comment about a component item in an assembly.

Examples

```
<ci idxnum="2" partnum="1238475" upa="1">Wheel assembly, front</ci>
<compcmt>For detailed breakdown, see <xref refid="wheelxmp">.</compcmt>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 23, “Creating parts catalog lists” on page 215.

Contexts

Children: text (#pcdata), Address, APL, Bin, Char, Cit, Date, Dec, Formula, Hex, L, MD, MV, Num, Oct, Ph, PK, PV, Q, RefKey, SynPh, Term, TM, XPh, XRef.

Parents: CompL.

CompL (component list)

Purpose

The CompL element contains a component list for an assembly.

Examples

```
<partasm id="bike" style="bkm:(layout=same)"><title>
Bicycle</title><mmobj><objref obj="bike">
<textalt>Bicycle</textalt>
</mmobj><compl>
<ci idxnum="1" partnum="4563423" upa="1">Bike</ci>
<compl>
<ci idxnum="1" partnum="1230987" upa="1">Frame</ci>
<ci idxnum="2" partnum="1238475" upa="1">Wheel assembly, front</ci>
<compcmt>For detailed breakdown, see <xref refid="wheelxmp">.</compcmt>
<ci idxnum="3" partnum="1234939" upa="1">Wheel assembly, rear</ci>
</compl>
</compl>
</partasm>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 23, “Creating parts catalog lists” on page 215.

Contexts

Children: CI, CompCmt, CompL.

Parents: CompL, PartAsm, PartAsmSeg.

Cond (procedure result)

Purpose

The Cond element is half of the Condition/Action pair of elements that is used in the DecisionPnt element. The Cond element contains a description of a condition that requires that some action be taken by the person following the procedure.

Examples

```
<PROC ID="BABYMAP" STYLE="BKM:(STYLE=BASE SEP=INLINE COMPACT)">
<TITLEBLK><TITLE>BABY JOHNNY IS CRYING</TITLE></TITLEBLK>
<PROCENTRY>SIX-MONTH OLD BABY JOHNNY WAS SLEEPING
PEACEFULLY; SUDDENLY HE BEGAN TO CRY.</PROCENTRY>
<PROCSTEP>
<PROCCMND>
<DESC>CHECK JOHNNY'S DIAPER</DESC>
</PROCCMND>
<DECISIONPNT>
<COND>IS THE DIAPER WET?</COND>
<THEN><PROCSTEP><PROCCMND>
<DESC>CHANGE THE DIAPER.</DESC>
</PROCCMND><PROCEXIT>JOHNNY WAS UNCOMFORTABLE.</PROCEXIT>
</PROCSTEP>
</THEN>
<ELSE>
<DESC>CONTINUE AT <XREF REFID="HUNGRY">.</DESC>
</ELSE>
</DECISIONPNT>
</PROCSTEP><PROCSTEP ID="HUNGRY">
<DECISIONPNT>
<COND>IS JOHNNY HUNGRY?</COND>
<THEN><PROCSTEP><DECISIONPNT>
<COND>DOES JOHNNY HAVE TEETH?</COND>
<THEN><PROCSTEP><STEPNOTES><LI>JOHNNY CAN EAT SOLID
FOOD.</LI>
<LI>CONTINUE AT <XREF REFID="FROZSTK">.</LI>
</STEPNOTES></PROCSTEP>
</THEN>
<ELSE><PROCSTEP ID="BOTTLE"><PROCCMND>
<DESC>WARM A BOTTLE.</DESC>
</PROCCMND><PROCCMND>
<DESC>FEED JOHNNY.</DESC>
</PROCCMND><PROCEXIT>JOHNNY NEEDED A BOTTLE.</PROCEXIT>
</PROCSTEP>
</ELSE>
</DECISIONPNT></PROCSTEP>
</THEN>
<ELSE><PROCSTEP><PROCCMND>
<DESC>ROCK JOHNNY TO SLEEP.</DESC>
</PROCCMND><PROCEXIT>JOHNNY WAS SLEEPY.</PROCEXIT>
</PROCSTEP>
</ELSE>
</DECISIONPNT>
</PROCSTEP></PROC>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 22, “Creating maintenance analysis procedures” on page 207.

Contexts

Children: text (#pcdata), DL, Fig, L, MMObj, Note, NoteList, OL, P, PBlk, Ph, Table, Term, TM, UL.

Parents: DecisionPnt.

ContainedDocs (documents in IBMLibEntry and LibEntry)

Purpose

The ContainedDocs element is used within IBMLibEntry and LibEntry elements to contain IDs of IBMLibEntry and LibEntry elements.

Examples

```
<CONTAINEDDOCS BIBIDS="bk1 bk2 bk3 bk4 bk5">
```

Attributes

See “Common Element Attributes (large set)” on page 227.

ID=*element_id*
  Contains the ID of a BibEntry or LibEntry element.

Usage

For more information about the ContainedDocs element, see “Defining library entries” on page 143.

Contexts

Children: empty.

Parents: IBMLibEntry, LibEntry.

CopyR (copyrights)

Purpose

The CopyR element defines the copyright information that must be referenced. Use the COPYR attribute to reference the CopyR element.

Specify one CopyR for every copyright holder for the document or division. You must specify at least the copyright holder and the first date. The presentation of the copyright statement in the final document is a function of the output style.

Examples

```
<IBMIDDOC COPYR="ibmprimary">
   ⋮
<PROLOG>
 <COPYRDEFS>
  <COPYR ID="YR1994">&copyr; COPYRIGHT INTERNATIONAL BUSINESS
MACHINES CORPORATION 1994.  ALL RIGHTS RESERVED.
   <P>This text is added to the end of the generated notice.</P>
   <P>Note to U.S. Government Users -- Documentation related to
    restricted rights -- Use, duplication or disclosure is subject to
    restrictions set forth in GSA ADP Schedule Contract with
    IBM Corp.
   </P>
</COPYR>
 </COPYRDEFS>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

ID=*copyright_id*
The ID of the CopyR element. Contains the ID for CopyR element.

Usage

See "Using CopyRDefs" on page 91.

Contexts

Children: text (#pcdata), DL, Fig, L, MMObj, Note, NoteList, OL, P, PBlk, Ph, Table, Term, TM, UL.

Parents: CopyrDefs.

CopyRDefs (copyright definitions)

Purpose

The CopyRDefs element defines copyright attributions for a document or division. Copyright attributions define the intellectual property rights held in the information contained by the document.

Use CopyRDefs to contain the copyright ownership information for the document or division. Put copyrights at the highest level to which they apply. For example, the primary author of the document should always have a copyright attribution at the document level, but a portion of a document may be copyrighted by someone else.

In order for the copyright to apply, it must be referenced within the document.

Examples

```
<PROLOG>
<COPYRDEFS>
 <COPYR ID="ibmprimary">
  © COPYRIGHT INTERNATIONAL BUSINESS
  MACHINES CORPORATION 1994.  ALL RIGHTS RESERVED.
   <P>This text is added to the end of the generated notice.</P>
   <P>Note to U.S. Government Users -- Documentation related to
   restricted rights -- Use, duplication or disclosure is subject to
   restrictions set forth in GSA ADP Schedule Contract with
   IBM Corp.</P>
  </COPYR>
 </COPYR>
</COPYRDEFS>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

CopyR
Contains copyright attributions for the document or division.

Usage

See "Using CopyRDefs" on page 91.

Contexts

Children: Copyr.

Parents: DProlog, Prolog, SpecDProlog.

Corp (enterprise name and address)

Purpose

Corp contains CorpName and address pairs for use in contexts like Author, Approvers, and Owners where either a person or an enterprise could be meaningful.

Examples

```
<authors>
<author><corp>
<corpname>International Business Machines</corpname>
</corp></author>
</authors>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Using reader’s comment form (RCF)” on page 103.

Contexts

Children: Address, CorpName.

Parents: Approvers, Author, Maintainer, Owners.

CorpName (corporation name)

Purpose

The CorpName simply contains the otherwise unstructured name of an enterprise, such as a company, government agency, or non-profit organization.

Examples

```
<authors>
<author><corp>
<corpname>International Business Machines</corpname>
</corp></author>
</authors>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Using reader’s comment form (RCF)” on page 103.

CorpName (corporation name)

Contexts

Children: text (#pcdata).

Parents: Corp, Publisher.

CoverDef (cover definition)

Purpose

The CoverDef element contains elements that reference to the art used for the document's covers.

Examples

This example shows how to define the artwork for the front and back covers.

```
<!entity front1 system "front1.eps" ndata graphics>
<!entity back1  system "back1.eps" ndata graphics>
 ...
<prolog><ibmbibentry><doctitle><titleblk>
<title>Sample Cover</title>
</titleblk></doctitle>
<coverdef>
<frontcover><mmobj><objref obj="front1">
<textalt>System/X cover artwork</textalt>
</mmobj></frontcover>
<backcover><mmobj><objref obj="back1">
<textalt>System/X back cover artwork</textalt>
</mmobj></backcover>
</coverdef>
</ibmbibentry></prolog>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See "Adding to the front or back cover (CoverDef)" on page 91.

Contexts

Children: BackCover, FrontCover, MMObj.

Parents: IBMBibEntry.

CritDate (critical date for a document)

Purpose

The CritDate element a date for the document. This currently does not reset the date symbol; the draft title page always shows the current date.

Examples

```
<CRITDATE>
<DATE>12 June 95</DATE>
<DESC>The date the document was approved for publication.</DESC>
</CRITDATE>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Date” on page 90.

Contexts

Children: Date, Desc.

Parents: CritDates.

CritDates (set of critical dates)

Purpose

The CritDates element contains a date in the life of the document.

Examples

```
<CRITDATES>
<CRITDATE>
<DATE>12 June 94
<DESC>The date the document was approved for publication.</DESC>
</CRITDATE>
</CRITDATES>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Date” on page 90.

Contexts

Children: CritDate.

Parents: DProlog, Prolog, SpecDProlog.

D (hierarchical division)

Purpose

The D (division) element defines the hierarchical organization of the information.

Migration Note

D replaces all the H*x* elements from BookMaster.

You must explicitly end the D element in order to start another D at the same hierarchical level. This is because the hierarchical level of each division is defined by the containment structure, not by explicit tag names.

Examples

A simple markup example of the D element is:

D (hierarchical division)

```
<d>
<dprolog><titleblk>
<title>About Hierarchical Divisions</title>
</titleblk></dprolog>
<dbody>
<p>Hierarchical divisions define the logical organization
of a document.</p>
</dbody></d>
```

Attributes

| **CHAPTERNUM=***chapter-number*
| You can use the CHAPTERNUM attribute on any first-level division tag to
| assign the chapter number or appendic letter. For example, this markup:

```
<d chapternum="5"><dprolog><titleblk><title>End of the line
```

| would cause the chapter number for the “End of the line” chapter to be 5. In
| an appendix, the appendic letter would be E.

COPYR=*copyrinf*
IBMCOPYR=*ibmcopyrinf*
Specifies the IBM or non_IBM copyright information.

IBMSEC=*sec_level*
SEC=*sec_level*
Specifies the security classification of the D content.

Toc=toc | notoc
Specifies whether the heading should appear in the table of contents. The default is for headings to appear in the table of contents; to a certain level (such as heading level 3). This replaces the older style=hidden attribute. Use this to prevent headings from appearing in the table of contents. It cannot be used to add a heading to the table of contents that would not normally appear. The levels of headings that appear in the table of contents are determined by the MaxToc attribute of the IBMIDDoc tag.

LANGUAGE=*lang_name*
Specifies the language in which the division is written.

| **LAYOUT=Document-Layout | OneCol | TwoCol | OffsetCol**
| Specifies the column-style for this portion of the book.

| **Document-Layout**
| The section uses the default layout for the document style.

| **OffsetCol**
| Formats the text using an indented, one-column layout. The headings
| format across the page, with the offset text, or indented from the
| margin.

| **OneCol**
| The text formats across the entire page.

| **TwoCol**
| The text formats in two columns.

| **STARTPAGE=***starting-page-number*
| The STARTPAGE (starting page) attribute allows you to assign the beginning
| page number to a section by using the that attribute on its division tag. It can
| be used with all first-level division tags. The STARTPAGE attribute value can
| be any positive integer, starting with 1. For example, if you use the following
| markup:

```
<d startpage="101"><dprolog><titleblk><title>Help information
...
<d startpage="201"><dprolog><titleblk><title>Safety information
```

the first chapter “Help information” starts on page 101, and the next chapter “Safety information” starts on page 201.

style=bkm:(*BookManager overrides***)**
IBMIDDoc and IDWB support BookManager override attributes that you can use when building books.

TopicID
You can substitute a topic identifier for the one that is automatically generated for BookManager books by using the TOPICID attribute on the heading or implied heading tag whose topic identifier you want to change. Here’s how:

```
<d style="bkm:(topicid='Contents-1')">
<dprolog><titleblk>
<title>Contents of Part 1</title>
</titleblk></dprolog>
<dbody></dbody></d>
```

The value of TOPICID must be a string of characters with no blanks.

TopicSel
You can prevent a heading or implied heading from being used as a topic by using the TOPICSEL attribute, like this:

```
<d style="bkm:(topicsel=no)">
<dprolog><titleblk>
<title>Contents of Part 1</title>
</titleblk></dprolog>
<dbody></dbody></d>
```

The value of TOPICSEL can be either YES or NO.

See “Common Element Attributes (large set)” on page 227.

Usage

See “Creating divisions (D element)” on page 20.

Contexts

Children: Abstract, DBody, DIntro, DProlog, DSum.

Parents: Appendix, BackM, Body, DBlk, DBody, DIntro, DSum, FrontM, LEDI, MsgItem, ProcIntro.

Danger (danger notice)

Purpose

Use Danger to create a danger notice, consisting of one or more paragraphs or other paragraph or phrase-level elements. Danger elements are normally used to contain warnings about actions that may cause injury or death to a person.

Examples

```
<danger>
Working under an automobile supported only
by the jack may result in injury or death. Always
use jack stands or ramps in axle pairs to support
your vehicle.
</danger>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “The perils of processing: Attention, caution, and danger” on page 48.

Contexts

Children: text (#pcdata), Address, Annot, APL, Bin, Bridge, CGraphic, Char, Cit, Date, Dec, DL, Fig, Formula, GL, Hex, L, Lines, LitData, LQ, MD, MkNote, MMObj, ModInfo, MV, Num, Oct, OL, P, ParmL, PBlk, Ph, PK, PV, Q, RefKey, Screen, SynPh, Syntax, Table, Term, TM, UL, Xmp, XPh, XRef.

Parents: AnnotBody, Bridge, DBody, Defn, DIntro, DSum, entry, LEDI, LI, LQ, ModDesc, ModItem, MsgItem, PBlk, ProcEntry, ProcIntro, Safety.

Date

Purpose

Use the Date element to contain a date. IBMIDDoc does not define the format of the date element, but processing systems can define format constraints for Date element content.

Examples

```
<ANNOT>
 <P>This change was made on <DATE>August 14, 1994</DATE>.
</ANNOT>
```

```
<ANNOT>
 <P>This document was formatted on <DATE SPEC="AUTO">.
</ANNOT>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

SPEC=AUTO
: Indicates that the presented date is to be defined by the presentation system and the defined document style. By default, this is the system date at the time the document is processed. When AUTO is specified, the element must be empty.

Usage

See “Date” on page 90.

Contexts

Children: text (#pcdata).

Parents: AnnotBody, Attention, Bridge, Caution, CompCmt, CritDate, Danger, Defn, Desc, entry, Fn, L, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, NoteBody, P, Ph, Q, Rev, SynNote, Warning.

DBlk (Division block)

Purpose

The DBlk element is used to organize divisions. A common use is to include two or more divisions from an object library.

Examples

```
<objlib>
<objlibbody><dblk id="somechapters">
<d>
<dprolog><titleblk>
<title>A heading</title>
</titleblk></dprolog>
<dbody>
<p>some interesting information</p>
</dbody></d>
<d>
<dprolog><titleblk>
<title>Another heading</title>
</titleblk></dprolog>
<dbody>
<p>more interesting information.</p>
</dbody></d>
</dblk></objlibbody>
</objlib>
 ...
<dblk conloc="somechapters" props="novice">
```

Attributes

Conloc
: The CONLOC attribute specifies that the content of another element of the same type is to be used as the content of the referencing element. This enables reuse of information. When the CONLOC attribute is specified, you cannot specify the element’s end tag. The result of using CONLOC is exactly the same as if the element being referred to had occurred at that point in the document. See “Reusing elements from an object library” on page 191.

  Attributes on the element are now passed to the element with the CONLOC; this is a feature that began with IDWB release 3.4, patch IDWXF036.

ID=*identifier*
: The ID attribute identifies an element within an SGML document. IDs must be unique within a single document. Any element that has an ID can be cross-referenced or linked to. IDs can be up to 64 characters long. IDs must start with an alphabetic character and can contain letters, numbers, dashes (-), or periods (.).

Props=*properties*
: The Props attribute is used to specify the condition under which the information contained within the element appears. See “Property-Based Retrieval” on page 195.

PropSrc
Points to an element whose properties are to be used as the properties of the referencing element. See Chapter 20, “Property and Class Definitions” on page 201.

Rev
The REV attribute references the Rev element which describes the last revision level of the information. See “Using Revisions” on page 109.

Status
ignored by processes

Contexts

Children: D.

Parents: Appendix, BackM, Body, DBody, FrontM, LEDI, MsgItem.

DBody (division body)

Purpose

DBody contains the content of a hierarchical division; that is, the main content of a chapter or topic.

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Creating divisions (D element)” on page 20.

Examples

```
<d>
<dprolog><titleblk>
<title>About Hierarchical Divisions</title>
</titleblk></dprolog>
<dbody>
<p>Hierarchical divisions define the logical organization
of a document.</p>
</dbody></d>
```

Contexts

Children: Annot, AsmList, Attention, BibList, Bridge, Caution, CGraphic, D, Danger, DBlk, DL, Fig, FnList, GL, L, LERS, Lines, LitData, LQ, MarkList, MkNote, MMObj, ModInfo, MsgList, Note, NoteList, OL, P, ParmL, PartAsm, PBlk, Proc, Screen, Syntax, Table, UL, Xmp.

Parents: Abbrev, Abstract, Bibliog, D, Glossary, Legend, Part, Preface, SOA.

Dec (decimal number)

Purpose

Use the Dec element to identify decimal data, which is data that is encoded in a base-10 numbering system.

Examples

```
<BIN>11000001</BIN> = <DEC>193<DEC>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See Table 1 on page 44.

Contexts

Children: text (#pcdata).

Parents: AnnotBody, Attention, Bridge, Caution, CompCmt, Danger, Defn, Desc, entry, Fn, L, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, MsgText, NoteBody, P, Ph, Q, SynNote, Term, Warning.

DecisionPnt (decision point)

Purpose

The DecisionPnt element defines one or more condition-action pairs that define the next step in a procedure.

Attributes

See “Common Element Attributes (large set)” on page 227.

Cond
Defines the condition which, if satisfied, indicates the action that should be taken.

Then
What action to take if the condition is satisfied.

Else
What action to take if the condition is note satisfied.

Usage

See Chapter 22, “Creating maintenance analysis procedures” on page 207.

Examples

```
<PROC ID="BABYMAP" STYLE="BKM:(STYLE=BASE SEP=INLINE COMPACT)">
<TITLEBLK><TITLE>BABY JOHNNY IS CRYING</TITLE></TITLEBLK>
<PROCENTRY>SIX-MONTH OLD BABY JOHNNY WAS SLEEPING
PEACEFULLY; SUDDENLY HE BEGAN TO CRY.</PROCENTRY>
<PROCSTEP>
<PROCCMND>
<DESC>CHECK JOHNNY'S DIAPER</DESC>
</PROCCMND>
<DECISIONPNT>
<COND>IS THE DIAPER WET?</COND>
<THEN><PROCSTEP><PROCCMND>
<DESC>CHANGE THE DIAPER.</DESC>
</PROCCMND><PROCEXIT>JOHNNY WAS UNCOMFORTABLE.</PROCEXIT>
</PROCSTEP>
</THEN>
<ELSE>
```

```
<DESC>CONTINUE AT <XREF REFID="HUNGRY">.</DESC>
</ELSE>
</DECISIONPNT>
</PROCSTEP><PROCSTEP ID="HUNGRY">
<DECISIONPNT>
<COND>IS JOHNNY HUNGRY?</COND>
<THEN><PROCSTEP><DECISIONPNT>
<COND>DOES JOHNNY HAVE TEETH?</COND>
<THEN><PROCSTEP><STEPNOTES><LI>JOHNNY CAN EAT SOLID
FOOD.</LI>
<LI>CONTINUE AT <XREF REFID="FROZSTK">.</LI>
</STEPNOTES></PROCSTEP>
</THEN>
<ELSE><PROCSTEP ID="BOTTLE"><PROCCMND>
<DESC>WARM A BOTTLE.</DESC>
</PROCCMND><PROCCMND>
<DESC>FEED JOHNNY.</DESC>
</PROCCMND><PROCEXIT>JOHNNY NEEDED A BOTTLE.</PROCEXIT>
</PROCSTEP>
</ELSE>
</DECISIONPNT></PROCSTEP>
</THEN>
<ELSE><PROCSTEP><PROCCMND>
<DESC>ROCK JOHNNY TO SLEEP.</DESC>
</PROCCMND><PROCEXIT>JOHNNY WAS SLEEPY.</PROCEXIT>
</PROCSTEP>
</ELSE>
</DECISIONPNT>
</PROCSTEP></PROC>
```

Contexts

Children: Cond, Else, Then.

Parents: ProcStep.

Defn (definition of a term)

Purpose

The Defn element contains the definition of a term.

Examples

```
<dl>
<dlentry><term>gopher</term>
<defn>A burrowing rodent that feeds on roots of plants.
</defn>
</dlentry>
<dlentry><term>lawn</term>
<defn>Gopher highway. <p>Can be identified by dinner-plate-sized
mounds of dirt where grass used to be.</p></defn>
</dlentry>
<dlentry><term>agapanthus</term>
<defn>Lovely flowering plant, the roots of which are
the preferred food of gophers. <p>If your flourishing
agapanthus suddenly keels over, it means a gopher
has had a feast.</p></defn>
</dlentry>
</dl>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See "Definition lists" on page 32.

Contexts

Children: text (#pcdata), Address, Annot, APL, Attention, Bin, Bridge, Caution, CGraphic, Char, Cit, Danger, Date, Dec, DL, Fig, Formula, GL, Hex, L, Lines, LitData, LQ, MD, MkNote, MMObj, ModInfo, MV, Note, NoteList, Num, Oct, OL, P, ParmL, PBlk, Ph, PK, PV, Q, RefKey, Screen, SynPh, Syntax, Table, Term, TM, UL, Xmp, XPh, XRef.

Parents: DLEntry, GLEntry, Parm.

DefnHd (definition description heading)

Purpose

The DefnHd element contains the heading for the description portion of a definition or parameter list.

Examples

```
<dl><termhd>Setting</termhd>
<defnhd>Description</defnhd>
<dlentry><term>Low</term>
<defn>A good setting for simmering soups.</defn>
</dlentry>
<dlentry><term>Medium</term>
<defn>After the water has boiled, use this setting
for cooking the spaghetti.</defn>
</dlentry>
<dlentry><term>High</term>
<defn>Use this setting to get water boiling fast.
</defn>
</dlentry>
</dl>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See "Definition lists" on page 32.

Contexts

Children: text (#pcdata), L, Ph, Term, TM.

Parents: DL, ParmL.

Delim (syntax delimiter)

Purpose

The Delim tag specifies a delimiter that is to indicate the start or end of keywords, variables, operators, or groups. The delimiter can consist of one or more characters.

Examples

```
<syntax>
<group>
<kwd>FRED</kwd>
<delim>+</delim>
<kwd>WILMA</kwd>
</group>
</syntax>
```

Attributes

OPTREQ=REQ | OPT
Indicates whether or not the delimiter is optional or required. REQ (required) is the default.

STARTEND=START | END
The STARTEND attribute has a value of START or END, depending upon whether the Delim element is the starting or ending delimiter in the syntax.

CONVAR=CONSTANT|VAR
Indicates whether the content of the element is a constant or a variable in the context where is it used.

LINKEND=*reference-id*
Contains an ID reference that enables a link to an arbitrary location in the document.

See "Common Element Attributes (large set)" on page 227.

Usage

See "The Delim (delimiter) element" on page 154.

Contexts

Children: text (#pcdata).

Parents: Group, SynPh.

Desc (element description)

Purpose

The Desc element contains a description of an element. Many elements allow Desc in their content, usually as the first element, or following the title. Desc is intended to contain a description of the content of the element within which Desc appears. The presentation effect of a given Desc element is determined by the style of the element that contains the Desc. For example, within a figure or table, the Desc may be presented as part of the caption; in another element the Desc may not be presented at all.

Examples

```
<fig ID="figa">
<cap>Your CPU with the Whantoozler 3.0 Installed</cap>
<desc>This figure shows the elegance of the Whantoozler when properly
installed.</desc>
</FIG>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Contexts

Children: text (#pcdata), Address, APL, Bin, Char, Cit, Date, Dec, DL, Fig, Hex, L, MD, MMObj, MV, Note, NoteList, Num, Oct, OL, P, PBlk, Ph, PK, PV, Q, RefKey, StepRef, SynPh, Term, TM, UL, XPh, XRef.

Parents: Authors, BibEntry, CritDate, Else, Fig, IBMBibEntry, IBMLibEntry, LERSDef, LibEntry, Mark, ModInfo, ModInfoDef, ModItemDef, MsgItemDef, ObjLib, Proc, ProcCmnd, PropDef, PropDesc, PropGroup, Qualif, Rev, Table, Then.

DIntro (division introduction)

Purpose

The DIntro element contains the introduction to a hierarchical division D.

Note that regular D elements are used to subdivide the DIntro section, but that they are not numbered like the divisions in the division body. This is possible because the divisions within the division introduction are structurally distinguished from the divisions in the division body.

Examples

```
<d>
<dprolog><titleblk>
<title>My Little Chapter</title>
</titleblk></dprolog>
<dintro>
<p>My little division introductory sentence.</p>
</dintro>
<dbody>
<p>Here's the beginning of my chapter.</p>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Division introductions” on page 24.

Contexts

Children: Annot, AsmList, Attention, BibList, Bridge, Caution, CGraphic, D, Danger, DL, Fig, FnList, GL, L, Lines, LitData, LQ, MarkList, MkNote, MMObj, ModInfo, Note, NoteList, OL, P, ParmL, PBlk, Screen, Syntax, Table, TitleBlk, TOC, UL, Xmp.

Parents: Abbrev, Abstract, Bibliog, D, Glossary, Legend, MasterIndex, Part, Preface, SOA.

DL (definition list)

Purpose

The DL element contains a list of pairs of terms and definitions. Use definition lists as a generic definition structure for defining things other than glossary terms. Entries can be organized within a definition list using DLBlk elements. Bridge elements can also be used to create transitions or connections between blocks of entries.

Examples

```
<dl>
<dlentry><term>gopher</term>
<defn>A burrowing rodent that feeds on roots of plants.
</defn>
</dlentry>
<dlentry><term>lawn</term>
<defn>Gopher highway. <p>Can be identified by dinner-plate-sized
mounds of dirt where grass used to be.</p></defn>
</dlentry>
<dlentry><term>agapanthus</term>
<defn>Lovely flowering plant, the roots of which are
the preferred food of gophers. <p>If your flourishing
agapanthus suddenly keels over, it means a gopher
has had a feast.</p></defn>
</dlentry>
</dl>
```

Attributes

TermWidth= Small | Meduim | Large | 1 | 2
You can use the TERMWIDTH attribute to determine the indentation size of the definition list. The valid choices are: small (.5 inch, the default), medium (1 inch), and large (2 inches). The value “1” is for 1-character width; “2” is for a 2-character width.

LINESPACE=SPACE | COMPACT
Specifies whether the items in the list should be compacted or have space between the items. Nested lists automatically inherit the setting of the outer list, but can override that default with their own setting.

| **DEF=***definition-name*
| Specifies the definition to be used. This points to a DefName attribute on a
| corresponding DEF tag. The settings defined on that DEF tag are used as
| defaults for this tag. See Chapter 9, “Using definition tags” on page 105 for
| more information.

| **HeadStyle=base_h | italic_h | bold_h | monospaced_h | underlined_h |
| bold_italic_h | italic_underlined_h | bold_underlined_h |
| bold_italic_underlined_h**
| Specifies the highlighting to use for the list’s TermHd and DefnHd heading
| tags. The default is bold.

| All values work for Xyvision, XHTML, HTML, and BookMaster. For IPF or
| RTF, there are some inconsistencies due to the limitations of those formats:
| “monospaced” is ignored, and the default style (bold) is used for terms and
| headings. “bold_italic_underlined” does not exist in IPF or RTF; this is treated
| the same as bold_underlined. “base” works on everything but definition terms;
| there is no plain style in IPF, so the default bold will not be over-ridden.

| **TermStyle=base | italic | bold | monospaced | underlined | bold_italic |**
| **italic_underlined | bold_underlined | bold_italic_underlined**
| Specifies the highlighting to use for the list's Term tags. The default is bold.

See "Common Element Attributes (large set)" on page 227.

Usage

See "Definition lists" on page 32.

Contexts

Children: Bridge, DefnHd, DLBlk, DLEntry, TermHd.

Parents: AnnotBody, Attention, BackCover, Bridge, Caution, Cond, Copyr, Danger, DBody, Defn, Desc, DIntro, DSum, EdNotices, entry, Fig, FigSeg, Fn, FrontCover, LEDesc, LEDI, LI, LQ, MkNote, ModDesc, ModItem, MsgItem, NItem, NoteBody, Notices, P, PBlk, ProcEntry, ProcExit, ProcIntro, Safety, Sem, SynNote, TextAlt, Warning.

DLBlk (definition list block)

Purpose

The DLBlk element is used to organize definition list entries within a definition list.

Examples

```
<dl>
<dlblk>
<dlentry><term>Cat</term>
<defn>A house pet
that purrs when happy.</defn></dlentry>
<dlentry><term>Dog</term>
<defn>A house pet that wags
its tail when happy.</defn></dlentry>
</dlblk>
<dlblk>
<dlentry><term>Fish</term>
<defn>A house pet
with scales that swims.</defn></dlentry>
<dlentry><term>Turtle</term>
<defn>A house pet with
scales that swims and walks slowly.</defn></dlentry>
</dlblk>
</dl>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See "Definition lists" on page 32.

Contexts

Children: Bridge, DLEntry, Title.

| Parents: DL.

DLDef (Definition list definition)

Purpose

The DLDef element sets attribute defaults for definition lists and parameter lists. DLDef goes within the document prolog to set definitions for the entire document; or within a division prolog to set definitions for just that division. The DLDef tag goes inside a PropDefs tag.

Examples

```
<propdefs>
<dldef defname="mega" termwidth="large" termstyle="bold_italic">
</propdefs>
...
<dl def="mega">
<dlentry><term>zebra</term>
<defn>Striped horsie.</defn>
</dlentry>
</dl>
```

Attributes

DEFNAME=*definition-name*
The DefName attribute identifies a definition element. DefName values must be unique within a single document. A element that has a DEF attribute can point a corresponding tag with a DefName attribute. DefName values can be up to 64 characters long. DefName values must start with an alphabetic character and can contain letters, numbers, dashes (-), or periods (.).

Definition tags can be specified without a DefName attribute in the document's prolog. This allows you to change the initial settings of attributes for the entire document when the DEF tag is in the document's prolog. In a division's prolog, it changes the initial settings for that division.

LINESPACE=SPACE | COMPACT
Specifies whether the items in the list should be compacted or have space between the items. Nested lists automatically inherit the setting of the outer list, but can override that default with their own setting.

TermWidth= Small | Meduim | Large | 1 | 2
You can use the TERMWIDTH attribute to determine the indentation size of the definition list. The valid choices are: small (.5 inch, the default), medium (1 inch), and large (2 inches). The value "1" is for 1-character width; "2" is for a 2-character width.

HeadStyle=base_h | italic_h | bold_h | monospaced_h | underlined_h | bold_italic_h | italic_underlined_h | bold_underlined_h | bold_italic_underlined_h
Specifies the highlighting to use for the list's TermHd and DefnHd heading tags. The default is bold.

All values work for Xyvision, XHTML, HTML, and BookMaster. For IPF or RTF, there are some inconsistencies due to the limitations of those formats: "monospaced" is ignored, and the default style (bold) is used for terms and headings. "bold_italic_underlined" does not exist in IPF or RTF; this is treated the same as bold_underlined. "base" works on everything but definition terms; there is no plain style in IPF, so the default bold will not be over-ridden.

TermStyle=base | italic | bold | monospaced | underlined | bold_italic | italic_underlined | bold_underlined | bold_italic_underlined
Specifies the highlighting to use for the list's Term tags. The default is bold.

Props=*properties*
The Props attribute is used to specify the condition under which the information contained within the element appears. See "Property-Based Retrieval" on page 195.

Usage

See Chapter 9, "Using definition tags" on page 105.

Contexts

Children: empty.

Parents: PropDefs, PropGroup.

DLEntry (definition list entry)

Purpose

The DLEntry element contains a single term and its definition. Use the DLEntry element within a definition list (DL) to define information that is not glossary information. Glossary entries (GLEntry) should be used for formal, dictionary-style definitions of words.

Examples

```
<dl>
<dlentry><term>gopher</term>
<defn>A burrowing rodent that feeds on roots of plants.
</defn>
</dlentry>
<dlentry><term>lawn</term>
<defn>Gopher highway. <p>Can be identified by dinner-plate-sized
mounds of dirt where grass used to be.</p></defn>
</dlentry>
<dlentry><term>agapanthus</term>
<defn>Lovely flowering plant, the roots of which are
the preferred food of gophers. <p>If your flourishing
agapanthus suddenly keels over, it means a gopher
has had a feast.</p></defn>
</dlentry>
</dl>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See "Definition lists" on page 32.

Contexts

Children: Defn, Term.

Parents: DL, DLBlk.

DocTitle (document title)

Purpose

The DocTitle element contains document title information. You should always provide a short title for IBM documentation. The short title will be used in places like bookshelf lists, citations, and the book's spine. The complete title that appears on the document title page or cover page is defined by the document style and may include data from other prolog elements.

Examples

```
<prolog>
<ibmbibentry><doctitle><titleblk>
<title>My Cute, Little Document</title>
</titleblk></doctitle>
<ibmdocnum>SC99-1234-01</ibmdocnum>
<authors>
<author><person>
<name>Fred Mertz</name>
<address>East Overshoe, SD</address>
</person></author>
</authors>
</ibmbibentry>
</prolog>
```

Usage

See "Document title" on page 88.

Contexts

Children: Library, TitleBlk.

Parents: BibEntry, IBMBibEntry.

DProlog (division prolog)

Purpose

The DProlog element contains metainformation about a division, which is information that describes the division, such as the division title, the author, and so on. It also contains many different types of markup definitions used to define classes and properties for the division.

Examples

```
<d>
<dprolog><titleblk>
<title>My Little Chapter</title>
</titleblk>
<revdefs>
<rev id="v3r4" ident="use">
<date>June 5th</date>
<desc>Something happened...</desc>
</rev>
</revdefs></dprolog>
<dbody>
<p>Here's the beginning of my chapter.</p>
<p rev="v3r4">Something that changed on June 5th.
</p>
</dbody></d>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Division prologs” on page 23.

Contexts

Children: Approvers, Authors, BibEntryDefs, CopyrDefs, CritDates, GlDefs, IBMProdInfo, IdxDefs, LDescs, Maintainer, MasterIndexInfo, MetaData, ObjLib, Owners, ProdInfo, PropDefs, QualifDefs, RetKey, RevDefs, TitleBlk.

Parents: D, Part.

DSum (division summary)

Purpose

DSum contains a summary of the informational content of the division.

Examples

```
<d>
<dprolog>
<titleblk><title>Wantoozler features</title>
</titleblk>
</dprolog>
<dbody>
 ...
</dbody>
<dsum>
<p>In summary, the Whantoozler can triple the
normal operating speed of your system.</p>
</dsum>
</d>
```

Usage

See “Creating divisions (D element)” on page 20.

Contexts

Children: Annot, AsmList, Attention, BibList, Bridge, Caution, CGraphic, D, Danger, DL, Fig, FnList, GL, L, Lines, LitData, LQ, MarkList, MkNote, MMObj, ModInfo, Note, NoteList, OL, P, ParmL, PBlk, Screen, Syntax, Table, TitleBlk, UL, Xmp.

Parents: Abbrev, Abstract, Bibliog, D, Glossary, Legend, MasterIndex, Part, Preface, SOA.

DVCFObj (DVCF Migration Element)

Purpose

This element should only be temporarily used when migrating BookMaster DVCF coding to IBMIDDoc. Do not continue to use this; you will be warned with messages when your document is processed.

DVCFObj (DVCF Migration Element)

Contexts

Children: any element.

Parents:.

EdNotices (edition notices)

Purpose

The EdNotices element contains the edition notices for the document, including any legally required statements about intended use, updates, and the like.

Examples

```
<ibmiddoc ibmcopyr="1996, 1999">
 ...
<ednotices><title>First Edition (June 1997)</title>
<p>This edition applies to the IBMIDDoc language,
Version 4.2, and to all subsequent releases
and modifications until otherwise indicated in new
editions.</p>
</ednotices>
```

Attributes

SPEC=MAN
: Specifies whether the edition notice is generated automatically or manually. At this time, MAN is the only supported value.

See "Common Element Attributes (large set)" on page 227.

Usage

See "Notices and Edition notices" on page 98.

Contexts

Children: DL, Fig, L, MMObj, Note, NoteList, OL, P, PBlk, Table, Title, UL.

Parents: FrontM.

Else (other procedure path to follow)

Purpose

The Else element contains the step or steps to follow if the Then condition is not met.

Examples

Usage

See Chapter 22, "Creating maintenance analysis procedures" on page 207.

Contexts

Children: Desc, Proc, ProcStep.

Parents: DecisionPnt.

Entry (table entry)

Purpose

The Entry element contains an entry within a row.

Examples

```
<table pgwide="0" id="tablesample">
<cap>Sample table caption</cap>
<tgroup cols="1">
<colspec colname="col1">
<tbody>
<row>
<entry colname="col1">my little</entry>
</row>
<row>
<entry colname="col1">sample table</entry>
</row>
</tbody>
</tgroup>
</table>
```

Attributes

COLNAME=*column_name*
Specifies the column name to which the Entry belongs.

NAMEST=*start_name*
Specifies the name of the leftmost column of a horizontal span.

NAMEEND=*end_name*
Specifies the name of the rightmost column of a horizontal span.

SPANNAME=*span_name*
Specifies the name of a horizontal span in a TGroup.

MOREROWS=*number*
Specifies the number of additional rows to add in a vertical span.

VALIGN=TOP|MIDDLE|BOTTOM
This attribute specifies the vertical alignment of the text contained in the Entry elements.

TOP
specifies alignment of the text at the top of the Entry elements.

MIDDLE
specifies alignment of the text at the middle of the Entry elements.

BOTTOM
specifies alignment of the text at the bottom of the Entry elements (the default).

ALIGN=LEFT | RIGHT | CENTER | JUSTIFY | CHAR
This attribute specifies the horizontal positioning of the text contained in the column:

LEFT
specifies left alignment (the default).

RIGHT
specifies right alignment.

CENTER
Specifies center alignment.

CHAR
Specifies the character that is used for alignment.

CHAROFF=*number*
Specifies the character offset for Entry elements in this column.

COLSEP=0 (NO) |1 (YES)
This attribute's value specifies that the internal column rules should be:

- drawn to the right of each Entry element that ends a column (1)
- not displayed at all (0)

ROWSEP=0 (NO) |1 (YES)
This attribute's value specifies that the internal row rules should be:

- drawn below each Entry element that ends a row (1)
- not displayed at all (0)

ROTATE=0 (NO) |1 (YES)
Specifies whether the entry should be rotated.

Shade=NOShade | XLight | Light | Meduim | Dark | XDark
Use the Shade attribute to specify the shading. This table example shows the shading values used in table cells:

Table 21. Cell entry shading. In the editor, use the modify attributes icon or Ctrl-A to set the Shade attribute for the cell's Entry tag.

noshade (0%)	xlight (5%)	light (26%)	medium (50%)	dark (74%)	xdark (100%)
the	quick brown	fox	jumps over	the lazy	

Usage

See Chapter 7, "Creating IBMIDDoc Tables" on page 67.

Contexts

Children: text (#pcdata), Address, Annot, APL, Attention, Bin, Bridge, Caution, CGraphic, Char, Cit, Danger, Date, Dec, DL, Fig, Formula, GL, Hex, L, Lines, LitData, LQ, MD, MkNote, MMObj, MV, Note, NoteList, Num, Oct, OL, P, ParmL, PBlk, Ph, PK, PV, Q, RefKey, Screen, SynPh, Syntax, Term, TM, UL, Xmp, XPh, XRef.

Parents: row.

ExternalFileName

Purpose

This specifies the file name of this document. This is used for PDF cross-referencing.

Examples

```
<externalfilename>iddugref</externalfilename>
```

Contexts

Children: text (#pcdata), Ph.

Parents: BibEntry, IBMBibEntry.

Fig (figure)

Purpose

The Fig element contains and identifies figures, such as images, examples, and formulas. The figure serves to contain the exhibits and associate a caption and a description with them. It also allows those exhibits to be referenced.

Use FigSeg elements to break long figures into smaller chunks to enable breaking of figures at logical points. In a code sample, for example, you can use one FigSeg for each subroutine to ensure that no subroutines are broken in the middle.

Examples

```
<fig style="bkm:(place=inline width=column)">
<lines>Here are some lines
in the sample, simple figure.</lines>
</fig>
```

Attributes

FRAME=NONE | BOX | RULES
Causes the figure to have a frame. The default is none — no frame.

Box Causes a box to be placed around the figure.

Rules Causes a line to be placed above and below the figure; to visually separate it from the surrounding text.

| **PGWIDE=0 | 1**
| This specifies the width of the figure. 1 is for a page-wide figure; 0 uses the
| current column width (0 is the default).

| **ScalePct=***scale-percent*
| You can use the ScalePct attribute to scale the text up or down in the element.
| The scale-percent is a positive, whole number. 100 is the normal size; 50 is 50%
| smaller; 200 is 200% larger. For example, this scales the text to 80% of the body
| text:

```
scalepct="80"
```

STYLE="bkm:(place=inline)"
This style override ensures the following:

place=inline
This causes the figure to be placed inline for BookMaster. Normally, BookMaster formats figures by floating them to the top of the next page.

| **DEF=***definition-name*
| Specifies the definition to be used. This points to a DefName attribute on a
| corresponding DEF tag. The settings defined on that DEF tag are used as
| defaults for this tag. See Chapter 9, "Using definition tags" on page 105 for
| more information.

See "Common Element Attributes (large set)" on page 227.

Usage

See “Figures” on page 57.

Contexts

Children: Annot, Bridge, Cap, CGraphic, Desc, DL, FigSeg, Formula, GL, L, Lines, LitData, LQ, MkNote, MMObj, Note, NoteList, OL, P, ParmL, PBlk, RefKey, Screen, Syntax, UL, Xmp.

Parents: AnnotBody, Attention, Bridge, Caution, Cond, Copyr, Danger, DBody, Defn, Desc, DIntro, DSum, EdNotices, entry, Fn, LEDesc, LEDI, LI, LQ, MkNote, ModDesc, ModItem, MsgItem, NItem, NoteBody, Notices, P, PBlk, ProcEntry, ProcExit, ProcIntro, Safety, Sem, SynNote, Warning.

FigDef (Figure definition)

Purpose

The FigDef element sets attribute defaults for figures. FigDef goes within the document prolog to set definitions for the entire document; or within a division prolog to set definitions for just that division. The FigDef tag goes inside a PropDefs tag.

Examples

```
<propdefs>
<figdef defname=colfigs pgwide=0 frame=box scalepct=120>
<propdefs>
...
<fig id=jkl def=colfigs>
```

Attributes

DEFNAME=*definition-name*
The DefName attribute identifies a definition element. DefName values must be unique within a single document. A element that has a DEF attribute can point a corresponding tag with a DefName attribute. DefName values can be up to 64 characters long. DefName values must start with an alphabetic character and can contain letters, numbers, dashes (-), or periods (.).

Definition tags can be specified without a DefName attribute in the document’s prolog. This allows you to change the initial settings of attributes for the entire document when the DEF tag is in the document’s prolog. In a division’s prolog, it changes the initial settings for that division.

FRAME=NONE | BOX | RULES
Causes the figure to have a frame. The default is none — no frame.

Box Causes a box to be placed around the figure.

Rules Causes a line to be placed above and below the figure; to visually separate it from the surrounding text.

PGWIDE=0 | 1
This specifies the width of the figure. 1 is for a page-wide figure; 0 uses the current column width (0 is the default).

ScalePct=*scale-percent*
You can use the ScalePct attribute to scale the text up or down in the element.

The scale-percent is a positive, whole number. 100 is the normal size; 50 is 50% smaller; 200 is 200% larger. For example, this scales the text to 80% of the body text:

```
scalepct="80"
```

Props=*properties*
The Props attribute is used to specify the condition under which the information contained within the element appears. See "Property-Based Retrieval" on page 195.

Usage

See Chapter 9, "Using definition tags" on page 105.

Contexts

Children: empty.

Parents: PropDefs, PropGroup.

FigList (list of figures)

Purpose

The FigList element causes a figure list to be generated.

Examples

```
<figlist><gendtitle></figlist>
```

Attributes

SPEC= AUTO | MAN
Specifies that the content of the element is generated. If SPEC=AUTO is specified, a list of all figures in the document is generated.

LAYOUT=Default-Layout | OneCol | TwoCol
Specifies the column-style for the section.

Default-Layout
The section uses the default layout for the document style.

OneCol
The headings and text format across the entire page.

TwoCol
The text formats in two columns. Headings format across the page or with the two-column text.

Usage

See "List of figures" on page 100.

Contexts

Children: CLE, GendTitle, RetKey, TitleBlk.

Parents: FrontM.

FigSeg (figure segment)

Purpose

The FigSeg element organizes the content of a figure into logical segments. The primary intent of FigSeg is to contain parts of a figure that must be kept together when the figure is presented.

Use multiple figure segments to break long figures into smaller chunks to enable breaking of figures at logical points. In a code sample, for example, you might use one figure segment for each subroutine, ensuring that no subroutines will be broken in the middle.

Examples

```
<fig id="samplefigdesc" style="bkm:(place=inline width=column)">
<cap>Here's a sample figure with a caption and description
</cap>
<desc>This figure has a description. Note that descriptions
have punctuations like sentences.</desc>
<figseg>
<xmp>Here is the first part
of a coding example</xmp>
</figseg>
<figseg>
<xmp>Here is the second part
of a coding example</xmp>
</figseg>
</fig>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Contexts

Children: Annot, Bridge, CGraphic, DL, Formula, GL, L, Lines, LitData, LQ, MkNote, MMObj, Note, NoteList, OL, P, ParmL, PBlk, RefKey, Screen, Syntax, UL, Xmp.

Parents: Fig.

FileNum (file number)

Purpose

The FileNum element contains the file number of the document. File numbers are unique to the IBMBibEntry element, and are only used for IBM products.

Examples

```
<FileNum>
444-4444-44
</FileNum>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Document title” on page 88.

Contexts

Children: text (#pcdata), Ph.

Parents: IBMBibEntry.

Fn (footnote)

Purpose

Use Fn to annotate text with notes that are not appropriate for inclusion in-line or to indicate the source for facts or other material used in the text. Footnotes are associated with the content of the element containing the footnote.

Examples

```
<p>There's a footnote<fn>While some folks do not like
footnotes; they sometimes contain a nugget of priceless
lore. Did you know IBMIDDoc's grandmother was named
ISIL?</fn> around here somewhere.</p>
```

Attributes

refid=*id*
: Refers to another footnote identifier. If this attribute is specified, this footnote must be empty.

See “Common Element Attributes (large set)” on page 227.

Usage

See “Footnotes” on page 46.

Contexts

Children: text (#pcdata), Address, Annot, APL, Bin, Bridge, CGraphic, Char, Cit, Date, Dec, DL, Fig, Formula, GL, Hex, L, Lines, LitData, LQ, MD, MMObj, ModInfo, MV, Note, NoteList, Num, Oct, OL, P, ParmL, PBlk, Ph, PK, PV, Q, RefKey, Screen, SynPh, Syntax, Table, Term, TM, UL, Xmp, XPh, XRef.

Parents: FnList.

FNList (footnote list)

Purpose

The FnList element contains a list of footnotes.

Examples

```
<FNLIST spec="auto">
```

Attributes

SPEC=AUTO
Specifies that the content of the element is generated. If SPEC=AUTO is specified, all Fn elements in the document are presented.

See “Common Element Attributes (large set)” on page 227.

Contexts

Children: Fn.

Parents: DBody, DIntro, DSum, LEDI, MsgItem, PBlk, ProcIntro.

Formula (math formula)

Purpose

The Formula element contains or references a mathematical formula.

The NOTATION attribute must not be used when the OBJ attribute is used. If neither the OBJ or NOTATION attribute is specified, the SGML processor assumes that the inline formula data is encoded using the Script Mathematical Formula Formatter (SMFF) mathematics formula language. Although SMFF is implied if the notation type is not specified, the preferred IBMIDDoc form is to explicitly specify the NOTATION=SMFF, when applicable.

The content of Formula must be inspected and any occurrences of `</` must be modified to use a symbol name `< sl` to insure the formula element is not prematurely ended by an unintended end tag delimiter in the data.

Examples

The first example illustrates how the entity is encoded. The second example shows how to use a formula element that contains the `wave1` entity reference.

```
<!ENTITY wave1 PUBLIC '+//ISBN 0-933186::IBM//ENTITY formula//EN' SDATA SMFF >
```

...

```
<FORMULA ID="pwave1" OBJ="wave1">
```

The next example shows a formula element that contains a formula using SMFF notation.

```
<FORMULA NOTATION="smff">
integral from 0 to infinity of d x
</FORMULA>
```

Attributes

OBJ=*file-entity-name*
Refers to an external file that contains mathematical formula specifications. The attribute value is the name of a declared entity. This attribute is only used when the formula data is in an external file, and the Formula element must be empty when it is used. Note that the entity declaration must include the notation of the mathematical formula information.

When using the OBJ attribute, the formula element must be empty. The entity declaration that defines the entity referred to by this attribute must include the notation used to encode the mathematical formula information.

Notation=SMFF
Refers to the notation that is used to encode the mathematical formula data that is included inline in the document. It should be specified if the formula data is inline in the document. It should not be specified if the Object attribute is used to refer to an external entity containing the formula data.

Future Enhancement

At this time, only the SMFF is supported. Other formats will be supported at a later date.

See "Common Element Attributes (large set)" on page 227.

Contexts

Children: text (cdata).

Parents: AnnotBody, Attention, Bridge, Caution, CompCmt, Danger, Defn, entry, Fig, FigSeg, Fn, L, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, NoteBody, P, Ph, Q, SynNote, Term, Warning.

Fragment (syntax fragment)

Purpose

The Fragment element contains a labeled subpart of a syntax definition. Use syntax fragments to organize subparts of a large syntax definition that either appear multiple times or are recursively defined, or parts that are too complicated to appear in place. Fragments are referred to with the FragRef element.

Examples

```
<syntax>
<fragref><title>Common attributes</title></fragref>
<fragment><title>Common attributes</title>
<group optreq="opt" style="bkm:(composite)">
<kwd>ID</kwd>
<oper>=</oper>
<var>identifier</var>
</group>
<group optreq="opt" style="bkm:(composite)">
<kwd>STYLE</kwd>
<oper>=</oper>
<var>style stuff</var>
</group>
</fragment>
</syntax>
```

Attributes

LINKEND=*element_id*
The ID value of the element being linked to or the ID of a NameLoc element.

See "Common Element Attributes (large set)" on page 227.

Fragment (syntax fragment)

Usage

See "The FRAGMENT and FRAGREF (fragment reference) element" on page 155.

Contexts

Children: FragRef, Group, SynNote, Title.

Parents: SynBlk, Syntax.

FragRef (syntax fragment reference)

Purpose

The FragRef element provides a logical reference to a syntax definition fragment. Use fragment references to create symbolic references to subparts of a syntax definition or to abstract constructs that are not explicitly defined. For example, you can reduce the complexity of a definition by replacing complex subparts with fragment references with meaningful titles. You can also use fragment references to define recursive constructs or to refer to abstract constructs that are not explicitly defined.

Examples

```
<syntax>
<fragref><title>Common attributes</title></fragref>
<fragment><title>Common attributes</title>
<group optreq="opt" style="bkm:(composite)">
<kwd>ID</kwd>
<oper>=</oper>
<var>identifier</var>
</group>
<group optreq="opt" style="bkm:(composite)">
<kwd>STYLE</kwd>
<oper>=</oper>
<var>style stuff</var>
</group>
</fragment>
</syntax>
```

Attributes

OPTREQ=REQ | OPT
: Indicates whether the fragment is optional or required. REQ (required) is the default.

FRAGID=*fragment_ID*
: Refers to a Fragment element. When FRAGID is specified, it is an error to specify any content or the FragRef end tag.

See "Common Element Attributes (large set)" on page 227.

Usage

See "The FRAGMENT and FRAGREF (fragment reference) element" on page 155.

Contexts

Children: Title.

Parents: Fragment, Group, SynBlk, Syntax.

FrontCover

Purpose

The FrontCover element contains a reference to the art used for the document's front cover.

Examples

```
<ibmbibentry><doctitle><titleblk>
<title>My Document</title>
</titleblk></doctitle>
<ibmdocnum></ibmdocnum>
<coverdef><frontcover><mmobj><objref obj="front1">
<textalt></textalt>
</mmobj></frontcover><backcover><mmobj>stago.objref obj="back1">
<textalt></textalt>
</mmobj></backcover></coverdef></ibmbibentry>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See "Adding to the front or back cover (CoverDef)" on page 91.

Contexts

Children: BibList, CGraphic, DL, Lines, LitData, MMObj, OL, P, PBlk, Table, UL, Xmp.

Parents: CoverDef.

FrontM (front matter)

Purpose

The FrontM element contains the material that precedes the body of a document, such as the preface or table of contents.

Examples

```
<ibmiddoc>
<prolog>
 ...
</prolog>
<frontm style="display='tipage cover spine'">
 ...
</frontm>
<body>
 ...
</ibmiddoc>
```

Attributes

STYLE=″display=′tipage cover spine′ ″
Sets style items such as title page, covers, and spine. See "Front matter (FrontM)" on page 98 for the values to use.

See "Common Element Attributes (large set)" on page 227.

FrontM (front matter)

Usage

See “Front matter (FrontM)” on page 98.

Contexts

Children: Abbrev, Abstract, Bibliog, D, DBlk, EdNotices, FigList, Glossary, IBMSafety, Legend, Notices, Preface, RCF, Safety, SOA, TList, TOC.

Parents: IBMIDDoc.

GendTitle (default title specification)

Purpose

The GendTitle element causes the system default title for certain specialized divisions to be used at processing time. The GendTitle element has no content. The processing application and document style determine the title that will be generated.

Examples

```
<toc><gendtitle></toc>
```

Attributes

ID=*identifier*
The ID attribute identifies an element within an SGML document. IDs must be unique within a single document. Any element that has an ID can be cross-referenced or linked to. IDs can be up to 64 characters long. IDs must start with an alphabetic character and can contain letters, numbers, dashes (-), or periods (.).

Style
The Style attribute contains either a reference to a separate style specification for an element or a set of element-specific presentation style definitions when the processing system does not support separate styles for elements. Assigning a value to the STYLE attribute modifies how an element is presented when it is output.

HyTime
ignored by processes

InfoMast
A fixed attribute used to classify the element.

Contexts

Children: empty.

Parents: FigList, IBMSafety, Index, PNIndex, RCF, Safety, SpecDProlog, TList, TOC.

GL (glossary list)

Purpose

The GL element contains glossary entries. A glossary list contains one or more glossary entries (GLEntry), which in turn contain a term and one or more definitions. Entries can be organized within a glossary list using glossary block (GLBlk) elements. Bridge elements can also be used to create transitions or connections between blocks of entries.

Glossary entries can also be used within normal element content to be collected automatically, or placed within document or division prologs when the terms apply to an entire document or a to specific division.

Unlike definition list entries, glossary entries can associate multiple definitions with a single term.

Glossary lists are normally contained by a Glossary division in the BackM.

Examples

```
<backm>
 ...
<glossary>
<specdprolog><gendttitle></specdprolog>
<dbody>
  <gl>...</gl>
</dbody>
</glossary>
 ...
</backm>
```

Attributes

SPEC=AUTO
Specifies that the content of the element is generated.

LINESPACE=SPACE | COMPACT
Specifies whether the items in the list should be compacted or have space between the items. Nested lists automatically inherit the setting of the outer list, but can override that default with their own setting.

| **DEF=***definition-name*
| Specifies the definition to be used. This points to a DefName attribute on a corresponding DEF tag. The settings defined on that DEF tag are used as defaults for this tag. See Chapter 9, “Using definition tags” on page 105 for more information.

RETKEY=None | NoDup
Use the RetKey attribute to enable automatic running headings for glossary lists. NoDup indicates that the first and last non-blank glossary terms on the page are to be used. The two terms are joined together, separated by a bullet or other character, and the combined string is used as the retrieval subject for the page. If the first and last glossary terms on the page are the same, only the last glossary term is displayed in the running heading or footing. The values First and FirstLast can also be coded; but they are not currently supported.

If you code any explicit RetKey elements, they are honored and will appear. If you nest elements that can generate a running heading (for example, a MsgList inside Lers), only the outer active generated heading is used. That is, if you specified automated RetKey generation for LERS and MSGLIST, a Msgno inside Lers will not be used in the RetKey area. But if you had an explicit RetKey inside the Msg, then the RetKey is honored as an explicit override.

See “Common Element Attributes (large set)” on page 227.

GL (glossary list)

Usage

See Chapter 13, “Glossaries” on page 137.

Contexts

Children: Bridge, GLBlk, GLEntry.

Parents: AnnotBody, Attention, Bridge, Caution, Danger, DBody, Defn, DIntro, DSum, entry, Fig, FigSeg, Fn, LEDI, LI, LQ, MkNote, ModDesc, ModItem, MsgItem, NoteBody, P, PBlk, ProcIntro, SynNote, Warning.

GLBlk (glossary list block)

Purpose

The GLBlk element organizes glossary list entries within a glossary list. For example, you can use GLBlk to create logical subdivisions within a long glossary list.

Examples

```
<GL>
<GLBLK>
<TITLE>A</TITLE>
<GLENTRY>
<TERM>acopy</TERM>
<DEFN>A transaction program which provides a command line interface to
the APPC File Transfer Protocol (AFTP) facility.
</DEFN>
</GLENTRY>
<GLENTRY>
<TERM>advanced program-to-program communication (APPC)</TERM>
<DEFN>The general facility characterizing
the LU 6.2 architecture and its
various implementations in products.
</DEFN>
<DEFN>Sometimes used to refer to the LU 6.2
architecture and its product
implementations as a whole, or to an LU 6.2 product feature in
particular, such as an APPC application program interface.
</DEFN>
</GLENTRY>
</GLBLK>
</GL>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Separating letter groups in a glossary” on page 138.

Contexts

Children: Bridge, GLEntry, Title.

| Parents: GL.

GLDef (Glossary list definition)

Purpose

The GLDef element sets attribute defaults for glossary lists. GLDef goes within the document prolog to set definitions for the entire document; or within a division prolog to set definitions for just that division. The GLDef tag goes inside a PropDefs tag.

Examples

```
<propdefs>
<gldef retkey="nodup">
</propdefs>
```

Attributes

DEFNAME=*definition-name*
: The DefName attribute identifies a definition element. DefName values must be unique within a single document. A element that has a DEF attribute can point a corresponding tag with a DefName attribute. DefName values can be up to 64 characters long. DefName values must start with an alphabetic character and can contain letters, numbers, dashes (-), or periods (.).

  Definition tags can be specified without a DefName attribute in the document's prolog. This allows you to change the initial settings of attributes for the entire document when the DEF tag is in the document's prolog. In a division's prolog, it changes the initial settings for that division.

LINESPACE=SPACE | COMPACT
: Specifies whether the items in the list should be compacted or have space between the items. Nested lists automatically inherit the setting of the outer list, but can override that default with their own setting.

RETKEY=None | NoDup
: Use the RetKey attribute to enable automatic running headings for glossary lists. NoDup indicates that the first and last non-blank glossary terms on the page are to be used. The two terms are joined together, separated by a bullet or other character, and the combined string is used as the retrieval subject for the page. If the first and last glossary terms on the page are the same, only the last glossary term is displayed in the running heading or footing. The values First and FirstLast can also be coded; but they are not currently supported.

  If you code any explicit RetKey elements, they are honored and will appear. If you nest elements that can generate a running heading (for example, a MsgList inside Lers), only the outer active generated heading is used. That is, if you specified automated RetKey generation for LERS and MSGLIST, a Msgno inside Lers will not be used in the RetKey area. But if you had an explicit RetKey inside the Msg, then the RetKey is honored as an explicit override.

Props=*properties*
: The Props attribute is used to specify the condition under which the information contained within the element appears. See "Property-Based Retrieval" on page 195.

Usage

See Chapter 9, "Using definition tags" on page 105.

Contexts

| Children: empty.

| Parents: PropDefs, PropGroup.

GlDefs (glossary definitions)

Purpose

The GlDefs element contains GLEntrys that can be referred to from other places in the document or division.

Contexts

Children: GLEntry.

Parents: DProlog, Prolog, SpecDProlog.

GlEntry (glossary list entry)

Purpose

The GLEntry element contains a single term and its definition. Use the GLEntry element within a glossary list (GL) to define information.

Examples

```
<gl>
<glentry><term>gopher</term>
<defn>A burrowing rodent that feeds on roots of plants.
</defn>
</glentry>
<glentry><term>lawn</term>
<defn>Gopher highway. <p>Can be identified by dinner-plate-sized
mounds of dirt where grass used to be.</p></defn>
</glentry>
<glentry><term>agapanthus</term>
<defn>Lovely flowering plant, the roots of which are
the preferred food of gophers. <p>If your flourishing
agapanthus suddenly keels over, it means a gopher
has had a feast.</p></defn>
</glentry>
</gl>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Definition lists” on page 32.

Contexts

Children: Defn, Term.

Parents: GL, GLBlk, GlDefs.

Glossary

Purpose

The DL element contains a list of pairs of terms and definitions. Use definition lists as a generic definition structure for defining things other than glossary terms. Entries can be organized within a definition list using DLBlk elements. Bridge elements can also be used to create transitions or connections between blocks of entries.

Examples

```
<gl>
<glentry><term>gopher</term>
<defn>A burrowing rodent that feeds on roots of plants.
</defn>
</glentry>
<glentry><term>lawn</term>
<defn>Gopher highway. <p>Can be identified by dinner-plate-sized
mounds of dirt where grass used to be.</p></defn>
</glentry>
<glentry><term>agapanthus</term>
<defn>Lovely flowering plant, the roots of which are
the preferred food of gophers. <p>If your flourishing
agapanthus suddenly keels over, it means a gopher
has had a feast.</p></defn>
</glentry>
</gl>
```

Attributes

LAYOUT=Document-Layout | OneCol | TwoCol | OffsetCol
Specifies the column-style for this portion of the book.

Document-Layout
The section uses the default layout for the document style.

OffsetCol
Formats the text using an indented, one-column layout. The headings format across the page, with the offset text, or indented from the margin.

OneCol
The text formats across the entire page.

TwoCol
The text formats in two columns.

Toc=toc | notoc
Specifies whether the heading should appear in the table of contents. The default is for headings to appear in the table of contents; to a certain level (such as heading level 3). This replaces the older style=hidden attribute. Use this to prevent headings from appearing in the table of contents. It cannot be used to add a heading to the table of contents that would not normally appear. The levels of headings that appear in the table of contents are determined by the MaxToc attribute of the IBMIDDoc tag.

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 13, “Glossaries” on page 137.

Contexts

Children: DBody, DIntro, DSum, SpecDProlog.

Parents: BackM, FrontM.

Group

Purpose

The Group element defines the syntax group and lets you give the group a name in a Title element. The Title element enables the Group to be automatically fragmented if it is too large to fit the current area.

Examples

```
<syntax>
<group>
<kwd>FORM</kwd>
<kwd>PROC</kwd>
</group>
</syntax>
```

Attributes

REPID=*repeat-ID*
Specifies the group repeats. the REPID points to a REPSEP element in the same syntax diagram.

OPTREQ=DEF | REQ | OPT
Indicates whether or not the group is a default, optional, or required. REQ is assumed.

| **CHOICESEQ=CHOICE | COMPOSITE | SEQ**
| Indicates whether the items for this group are choices (you select one of the
| items), sequential (you enter each of them in order), or composite (sequential
| with no space or lines between the syntax elements).

See “Common Element Attributes (large set)” on page 227.

Usage

See “The Group element” on page 150.

Contexts

Children: Delim, FragRef, Group, Kwd, Oper, Sep, SynNote, Title, Var, XRef.

Parents: Fragment, Group, SynBlk, Syntax.

Hex (hexadecimal)

Purpose

Use the Hex element to identify hexadecimal data.

Examples

```
<BIN>1100 0001</BIN> = <HEX>C1</HEX>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See Table 1 on page 44.

Contexts

Children: text (#pcdata).

Parents: AnnotBody, Attention, Bridge, Caution, CompCmt, Danger, Defn, Desc, entry, Fn, L, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, MsgText, NoteBody, P, Ph, Q, SynNote, Term, Warning.

IBMBibEntry (IBM bibliographic entry)

Purpose

Use IBMBibEntry to define the bibliographic information about an IBM document.

IBMBibEntry elements can be specified in a BibEntryDefs element or object container and used by reference from within a document, for example, from Cit and Q elements. When IBMBibEntry is specified within Cit, Q, and LQ, the IBMBibEntry elements are collected for use in a generated bibliography.

Contained IBMBibEntry elements and IBMLibEntry elements are normally used within re-used information in order to ensure that the re-used information is completely self-contained. In other words, these elements should be used within the scope of the information that is being re-used. For example, if a Division element is re-used, the IBMBibEntry and IBMLibEntry elements should be contained within that same division's DProlog element. This allows these elements to be completely contained, and thus completely re-usable, along with the division that is being re-used.

Examples

```
<ibmiddoc>
<prolog><ibmbibentry><doctitle>
<library><titleblk>
<title>My Library</title>
</titleblk></library>
<titleblk>
<title>My Document of Interesting Things</title>
<stitle>Interesting things</stitle>
<subtitle>Or, Cool Things I Like</subtitle>
</titleblk></doctitle></ibmbibentry>
```

Attributes

DOCLINK=*ID*
: The DocLink attribute specifies the ID of the URL defined on a Notloc element.

DOCNAME=*entity_name*
: Contains a reference to the ID or name of an entity that is defined in the document that must also be referenced by a NameLoc element. This indicates a cross-document target with the specified ID value.

See "Common Element Attributes (large set)" on page 227.

IBMBibEntry (IBM bibliographic entry)

Usage

See Chapter 14, “Bibliographies and citations” on page 141.

Contexts

Children: Authors, CoverDef, Desc, DocTitle, ExternalFileName, FileNum, IBMDocNum, IBMPartNum, ISBN, OrigIBMDocNum, PrtLoc, PublicId, Publisher, RetKey, VolId.

Parents: BibEntryDefs, BibList, Cit, Prolog.

IBMBOFNum (bill of forms number)

Purpose

The IBMBOFNum element contains the IBM Bill of Forms number for the described library.

Examples

```
<bibentrydefs>
<ibmlibentry>
<library><titleblk><title>BS/300</title></titleblk>
</library>
<ibmbofnum>SBOF-1234-0</ibmbofnum>
</ibmlibentry>
</bibentrydefs>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Contexts

Children: text (#pcdata), Ph.

Parents: IBMLibEntry.

IBMDocNum (IBM document number)

Purpose

The IBMDocNum element contains the assigned IBM document number for the document.

Examples

```
<ibmbibentry><doctitle><titleblk>
<title>My Document</title>
</titleblk></doctitle>
<ibmdocnum>SC99-1234-00</ibmdocnum>
<ibmpartnum>1234F99</ibmpartnum>
</ibmbibentry><ibmprodinfo>
<prodname>My Product</prodname>
<ibmpgmnum>1234-XX1</ibmpgmnum>
<ibmfeatnum>1754</ibmfeatnum>
</ibmprodinfo>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Document number” on page 89.

Contexts

Children: text (#pcdata), Ph.

Parents: IBMBibEntry.

IBMFeatNum (IBM feature number)

Purpose

The IBMFeatNum element contains the assigned IBM feature number for the document.

Examples

```
<ibmbibentry><doctitle><titleblk>
<title>My Document</title>
</titleblk></doctitle>
<ibmdocnum>SC99-1234-00</ibmdocnum>
<ibmpartnum>1234F99</ibmpartnum>
</ibmbibentry><ibmprodinfo>
<prodname>My Product</prodname>
<ibmpgmnum>1234-XX1</ibmpgmnum>
<ibmfeatnum>1754</ibmfeatnum>
</ibmprodinfo>
```

Contexts

Children: text (#pcdata), Ph.

Parents: IBMProdInfo.

Attributes

See “Common Element Attributes (large set)” on page 227.

IBMIDDoc (IBM-specific product documentation)

Purpose

The IBMIDDoc element contains IBM product information. The IBMIDDoc document type is used within IBM to create information deliverables for IBM products. This document type conforms to the IBM InfoMast Architecture and the HyTime standard (ISO/IEC 10744). The IBMIDDoc element contains an entire IBMIDDoc document.

An IBMIDDoc document is divided into four main elements: Prolog, FrontM, Body, and BackM. Only the Body element is required. The Prolog contains all the information that describes the document itself, such as the document title, author, document numbers, property definitions, and so on. The other sections contain the content of the document organized into divisions, either by D elements or by specialized divisions, such as Preface or Bibliog.

Examples

```
<IBMIDDOC SEC="IXM Confidential" LANGUAGE="USENGLISH" "COPYR="ibmprimary" "lotus"
 "IBMCOPYR="1994, 1995" ID="edfl0mstv2r1">
 ...
</IBMIDDOC>
```

Attributes

In addition to support for general attributes, IBMIDDoc can also have several other attributes:

AppPrefix
: Controls the automatic generation of the word ″Appendix″ from the heading text.

  DEFAULT-APP
  : Use the default for this style. (This is the default value.)

  TEXT-APP
  : Add text and the number.

  NONE-APP
  : Do not add any prefix.

  NUMONLY-APP
  : Add the division number as a prefix.

ChapPrefix
: Controls the automatic generation of the word ″Chapter″ from the heading text.

  DEFAULT-CHAP
  : Use the default for this style. (This is the default value.)

  TEXT-CHAP
  : Add text and the number.

  NONE-CHAP
  : Do not add any prefix.

  NUMONLY-CHAP
  : Add the division number as a prefix.

Class
: Has a few values for HTML-published information. No entry generates a HTML-like book; running feet connect the pages together; headings link back to the table of contents.

  article
  : Specified HTML article format. A table of contents is generated. The articles have no automatic running footing that connect the pages together. Use this with the HTML frame option to generate a frame-based set of articles with a “twisty” table of contents.

  articles
  : Also specifies HTML article format. No table of contents is generated.

CLASSIF= CONFRES | RES | LIC
: Identifies the classification of restricted materials.

  CONFRES
  : Confidential restricted material

RES
Restricted material

LIC
Licensed material

CLASSIF=LIC for the style TIV8x11, causes that Tivoli style to include the licensed statement on each page and on the cover.

Copyr=*reference-ID*
References the ID for a copyright definition defined in an IBMBibEntry element.

DOCSTYLE=*document-style*
Specifies the style of the document. You can specify these styles:

IBM8X11
8-1/2 by 11 inch style. Replaces BookMaster style IBMXAGD.

IBM7X9
7 by 9 inch style. Replaces BookMaster style IBMXGGD.

IBM2COL
8.5x11 style (2 column layout)

IBMCD
4.75x4.75 style (for CD Jewel Case booklets)

IBMREFC
Reference cards (3-5/8x9in.).

IBM5X8
5.5x8.5 style (for hardware).

IBM4X6
4.25x6.25 style (for hardware).

IBM8X5
5.5x8.5 landscape style (for hardware).

IBM9X7
7x9 landscape style.

If you create a PDF from this style, the pages may switch between landscape and portrait presentation in Adobe Acrobat Reader or Exchange. Add the following lines to your PostScript file before distilling it to prevent this from occurring:

```
/currentdistillerparams where {pop}
{userdict /currentdistillerparams {1 dict} put} ifelse
/setdistillerparams where {pop}
{userdict /setdistillerparams {pop} put} ifelse
<< /AutoRotatePages /All >> setdistillerparams
```

IBMLAND
Printer System’s landscape books. (Not for BookMaster)

IBMXAGD
User Guides (8.5x11in., A4); old BookMaster style.

IBMXARF
Reference (8.5x11in., A4); old BookMaster style. This can be replaced by using a style of ibm8x11 and a layout of onecol.

IBMXGGD
Summary Guides (7-3/8x9in.); old BookMaster style.

TIV7X9
7x9 style for Tivoli

This style creates automatic running headers for titles. The style puts Part, Chapter, and Tivoli head 1 text in the RETKEY area. The STitle content, if specified, replaces the Title content in the running heading.

TIV8X11
8.5x11 style for Tivoli

This style creates automatic running headers for titles. The style puts Part, Chapter, and Tivoli head 1 text in the RETKEY area. The STitle content, if specified, replaces the Title content in the running heading.

OBIPORT
5.5x8.5 style (for Options by IBM)

OBIWWA6P
4.25x5.75 style (for Options by IBM)

SMALLFLG
3.625x8.5 style (for hardware)

IBMCopyr=*current-year* | *first-year, current-year*
Specifies the copyright date year for IBM publications. You enter either one date `1999` or two `1999, 2000`.

IBMSEC=UNC | IC
Specifies the IBM security classification for the document. Note that you should use the SEC attribute instead of IBMSEC, with the security classification typed out. If you specify both IBMSEC and SEC, the SEC attribute is used.

unc	Unclassified
ic	IBM Confidential

Language
Specifies the language in which the document is written.

The valid values for the Language attribute on IBMIDDoc element are:
- BDUTCH or nl_BE
- BFRENCH or fr_BE
- BPORTUGUESE or pt_BR
- BULGARIAN or bg_BG
- CATALAN or ca_ES
- CENGLISH or en_CA
- CFRENCH or fr_CA
- CROATIAN or hr_HR
- CZECH or cs_CZ
- DANISH or da_DK
- DUTCH or nl_NL
- ENGLISH, en_US, or USENGLISH
- ESTONIAN or et_EE
- FINNISH or fi_FI
- FRENCH or fr_FR
- GERMAN or de_DE
- GREEK or el_GR
- HUNGARIAN or hu_HU
- ICELANDIC or is_IS
- ITALIAN or it_IT
- JAPANESE or ja_JP
- KOREAN or ko_KR

- LATVIAN or lv_LV
- LITHUANIAN or lt_LT
- MACEDONIAN or mk_MK
- NORWEGIAN or no_NO
- POLISH or pl_PL
- PORTUGUESE or pt_PT
- ROMANIAN or ro_RO
- RUSSIAN or ru_RU
- SCHINESE or zh_CN
- SERBIAN or sr_SP
- SFRENCH or fr_CH
- SGERMAN or de_CH
- SITALIAN or it_CH
- SLOVAK or sk_SK
- SLOVENIAN or sl_SI
- SPANISH or es_ES
- SWEDISH or sv_SE
- TCHINESE or zh_TW
- THAI or th_TH
- TURKISH or tr_TR
- UKENGLISH or en_GB

LAYOUT=Document-Layout | OneCol | TwoCol | OffsetCol
Specifies the column-style for this portion of the book.

Document-Layout
The section uses the default layout for the document style.

OffsetCol
Formats the text using an indented, one-column layout. The headings format across the page, with the offset text, or indented from the margin.

OneCol
The text formats across the entire page.

TwoCol
The text formats in two columns.

MAXTOC=*number*
Specifies the maximum heading level to be included in the table of contents. The default for the style IBM8X11 is 3. Specifying `maxtoc=4` will include divisions to heading level 4.

MULTIVOL=<u>OneVol</u> | Index-Folio
This indicates whether the book is part of a multiple-volume set. Specifying "Index-Folio" adds "X-" as a prefix for the page numbers in the index and starts the page numbering from 1.

PageNumber
There are three options for the PageNumber attribute: FBC, SEQ, and Default-Folio.

SEQ
Sequential page numbering: The front matter page numbers use roman numerals. The body and back matter use arabic numerals.

Note: The 3.2 GA version of the Xyvision code will only support PageNumber=Seq.

FBC
: Folio-by-Chapter numbering: The front matter page numbers use roman numerals, body and appendixes add the chapter or appendix number as a prefix and restart the page number at every new level-1 division. For example, chapters and sections are numbered 1-1, 1-2, 2-1, 2-2, etc. Appendixes are numbered A-1, A-2, etc. Non-appendix back matter sections are numbered X-1, X-2, etc. and are not reset at new level-1 divisions.

Default-Folio
: This is the default value for PageNumber. Use the default page numbering style for this document style. Currently, all document styles default to SEQ.

PartPrefix
: Controls the automatic generation of the word "Part" in the heading text.

DEFAULT-PART
: Use the default for this style. (This is the default value.)

TEXT-PART
: Add text and the number.

NONE-PART
: Do not add any prefix.

NUMONLY-PART
: Add the division number as a prefix.

SEC
: Specifies the security classification. Note that you should use the SEC attribute instead of IBMSEC, with the security classification typed out. If you specify both IBMSEC and SEC, the SEC attribute is used.

STYLE=*overrides*
: This allows specific style overrides.

keepblanks and **removeblanks**
: These are used to control how blanks are treated. The default behavior keeps blanks at the end of phrase elements (such as Ph, Address, and Term), and literal data (such as Xmp, Cgraphic, and LitData). It will remove trailing blanks at the end of other elements, such as paragraphs (P). You can use the KEEPBLANKS option to keep all blanks; both those at the end of phrases and those at the end of paragraphs. You can use the REMOVEBLANKS option to remove all trailing blanks, both those at the end of phrases and those at the end of paragraphs.

As an example, the default for this markup is to keep the blanks inside the phrase tags. So this markup:

```
Hi <ph style="bold"> there </ph> handsome.
```

Effectively becomes this when the output is formatted:

Hi **there** handsome.

Through migration from BookMaster or other means, several times the phrase tags were done this way:

```
Hi <ph style="bold">there </ph>handsome.
```

If you use REMOVEBLANKS, the text concatenates like this:

Hi **there**handsome.

xpp:(bookmarks)
Causes the bookmarks in Acrobat PDF files to match the table of contents. This creates a most excellent way of navigating the PDF. This is now the default setting.

xpp:(justify)
For DBCS languages only, this causes the formatting for flowed text items to be left and right-justified. The opposite setting is `nojustify` or `ragged`.

MLSPrefix=YES | NO
Indicates the document is part of a multiple-language safety book. The page number prefixes are determined by the document's language attribute.

BRAND
Specifies the type of product identification branding to be used for the document.

DefaultBrand
This is the default; no special branding information is produced.

eserver-white
For the IBM @server brand, this specifies black cover text on a white background.

eserver-black
For the IBM @server brand, this specifies white cover text on a black background. Don't use this one without prior approval by SDF.

NewBrand
Indicates the value of the NEWBRAND attribute should be used.

NEWBRAND=*brand-name*
This is where you would enter a special brand name; as defined by the ID Workbench team, to handle a future brand in the middle of a release.

UNMSPACE=Separate | unify
Currently not used.

ID Allows you to assign an identifier to the entire document.

Company
Specifies the company for non-IBM trademarks.

DTDVersion
A fixed attribute that indicates the level of the DTD.

Contexts

Must occur in same entity (file) as the IBMIDDoc document type declaration (DTD) and must be highest-level element in the document.

Children: BackM, Body, FrontM, Prolog.

Parents:.

IBMLibEntry (IBM document library definition)

Purpose

The IBMLibEntry element contains an IBM-specific library definition. Library entries contain information about a library or collection of documents.

Use IBMLibEntry to define the bibliographic information about a library or other collection of IBM documents. You can use the Class attribute and the ClassDef element to define different classes of LibEntry to correspond to different classes of collection. For example, you may have product libraries that are themselves collected into larger libraries of libraries. You could define a class of "CollectionKit" for libraries of libraries and then use IBMLibEntry elements to define the contents of a given collection kit.

IBMLibEntry can also be specified in a BibEntryDefs section or ObjLib and used by reference from within a document, for example, from Cit elements.

When IBMLibEntry elements are specified within Cit. the IBMLibEntry elements are collected for use in a generated bibliography.

Contained IBMBibEntry elements and IBMLibEntry elements are normally specified within re-used information in order to ensure that the re-used information is completely self-contained. In other words, these elements should be used within the scope of the information that is being re-used. For example, if a Division element is re-used, the IBMBibEntry and IBMLibEntry elements should be contained within that same division's DProlog element. This allows these elements to be completely contained, and thus completely re-usable, within the division that is being re-used.

Examples

```
<IBMLIBENTRY ID=IBMIDDocLIB>
 <LIBRARY>IBMIDDoc Library</LIBRARY>
 <PUBLISHER>IBM Corporation
  <ADDRESS>
  </ADDRESS>
 </PUBLISHER>
 <PRTLOC>USA
 <IBMBOFNUM>SBOF-6000-00
 <PUBLICID>+//ISBN 0-19-9999
 //LIB SBOF-6000-00
 /IBMIDDoc Library
 /rickd@nando.net
 /Networking Software
 //EN
 <CONTAINEDDOCS BIBIDS="IBMIDDocUG IBMIDDocTUT">
 <DESC>Documentation for the IBMIDDoc language.
 IBMIDDoc is an SGML language for creating printed and online
 technical information.
</IBMLIBENTRY>
```

Attributes

Library
The library name.

Publisher
Contains the name of the publisher followed by an optional address element.

PrtLoc
Contains the name location where the document was printed.

IBMBOFNum
The IBM bill of forms (BOF) number assigned to this library.

IBMPartNum
The IBM part number assigned to this library.

ProdName
The product name with which this library is associated.

ISBN
The ISBN number assigned to this library.

PublicID
The SGML public identifier assigned to this library. This is the same public identifier used in entity declarations for the system object that represents this library.

ContainedDocs
Defines the documents contained in this library and the default order for presenting the documents when the library definition is presented.

The contained documents can be identified directly using Cit elements, or indirectly by specifying BibEntry or LibEntry IDs.

The order the contained documents are specified in ContainedDocs defines the default order for presenting the library contents.

Desc
Contains a description of the library.

Contexts

Children: ContainedDocs, Desc, IBMBOFNum, IBMPartNum, ISBN, Library, ProdName, PrtLoc, PublicId, Publisher.

Parents: BibEntryDefs, BibList, Cit.

IBMMail (IBMMail e-mail address)

Purpose

The IBMMail element contains an IBMMail email address.

Examples

```
<IBMIDDOC>
 <PROLOG>

   ⋮
  <OWNERS>
   <CORP>
    <CORPNAME>IBM CORPORATION</CORPNAME>
    <ADDRESS>INFORMATION DEVELOPMENT
              IBM RTP
              DEPT. E14D
              500/D162
              ATTN: Rick Dennis
     <INTERNET>rickd@nando.net</INTERNET>
     <PHONE>919-254-4062</PHONE>
     <VNET>RICKD@RTPNOTES</VNET>
     <IBMMAIL>IBMMail Exchange: USIBM3345 at IBMMAIL</VNET>
```

```
      <POSTALCODE>27614</POSTALCODE>
     </ADDRESS>
    </CORP>
   </OWNERS>
   <MAINTAINER>
    <PERSON>
     <NAME>Rick Dennis</NAME>
    </PERSON>
   </MAINTAINER>
  </PROLOG>
```

Attributes

#PCDATA
Contains the IBMMail email address.

Contexts

Children: text (#pcdata).

Parents: Address.

IBMPartNum (IBM part number)

Purpose

The IBMPartNum element contains the IBM part number of the document.

Examples

```
<ibmbibentry><doctitle><titleblk>
<title>My Document</title>
</titleblk></doctitle>
<ibmdocnum>SC99-1234-00</ibmdocnum>
<ibmpartnum>1234F99</ibmpartnum>
</ibmbibentry><ibmprodinfo>
<prodname>My Product</prodname>
<ibmpgmnum>1234-XX1</ibmpgmnum>
<ibmfeatnum>1754</ibmfeatnum>
</ibmprodinfo>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Contexts

Children: text (#pcdata), Ph.

Parents: IBMBibEntry, IBMLibEntry.

IBMPgmNum (IBM program number)

Purpose

The IBMPgmNum element contains the IBM program number of an IBM program product that is described by the document.

Examples

```
<ibmbibentry><doctitle><titleblk>
<title>My Document</title>
</titleblk></doctitle></ibmbibentry>
<ibmprodinfo>
<prodname>My Product</prodname>
<ibmpgmnum>1234-XX1</ibmpgmnum>
<ibmfeatnum>1754</ibmfeatnum>
</ibmprodinfo>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Using IBMProdInfo” on page 92.

Contexts

Children: text (#pcdata), Ph.

Parents: IBMProdInfo.

IBMProdInfo (IBM product information)

Purpose

The IBMProdInfo element contains information about an IBM product that is associated with the document.

Examples

```
<ibmbibentry><doctitle><titleblk>
<title>My Document</title>
</titleblk></doctitle></ibmbibentry>
<ibmprodinfo>
<prodname>My Product</prodname>
<ibmfeatnum>1754</ibmfeatnum>
</ibmprodinfo>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Using IBMProdInfo” on page 92.

Contexts

Children: IBMFeatNum, IBMPgmNum, ModLvl, ProdName, Release, Version.

Parents: DProlog, Prolog, SpecDProlog.

IBMSafety (IBM safety notices)

Purpose

The IBMSafety element is designed to contain IBM-specific safety notices about safe hardware practices. THis is not yet implemented.

Examples

```
<frontm>
<ibmsafety spec="auto"><gendtitle></ibmsafety>
 ...
</frontm>
```

Attributes

| **LAYOUT=Document-Layout | OneCol | TwoCol | OffsetCol**
| Specifies the column-style for this portion of the book.

| **Document-Layout**
| The section uses the default layout for the document style.

| **OffsetCol**
| Formats the text using an indented, one-column layout. The headings
| format across the page, with the offset text, or indented from the
| margin.

| **OneCol**
| The text formats across the entire page.

| **TwoCol**
| The text formats in two columns.

SPEC=AUTO|MAN
Specifies that the content of the element is generated. SPEC=AUTO is the default value, and causes the appropriate generated text to be included.

See “Common Element Attributes (large set)” on page 227.

Contexts

Children: GendTitle, RetKey, TitleBlk.

Parents: FrontM.

IdxDefs (central index entries)

Purpose

The IdxDefs element contains central index entries for the document or division.

Examples

```
<prolog><ibmbibentry><doctitle><titleblk>
<title>Index test</title>
</titleblk></doctitle></ibmbibentry>
<idxdefs>
<i1 id="becha"><idxterm>a bechamel sauce</idxterm></i1>
<i1 seeid="becha"><idxterm>a white sauce</idxterm></i1>
</idxdefs></prolog>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Defining index entries (central indexing)” on page 120.

Contexts

Children: I1, I2, I3, IRef.

Parents: DProlog, Prolog, SpecDProlog.

IdxTerm (index term)

Purpose

The IdxTerm element contains a term that is to be included in the index.

Examples

```
<i1><idxterm>dessert sauces</idxterm></i1>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See Chapter 11, "Indexing" on page 115.

Contexts

Children: text (#pcdata), Ph.

Parents: ClassDef, I1, I2, I3.

Index

Purpose

The Index element contains a title for the index, if one is specified.

The normal use of Index is to contain an index that is automatically generated from the index entries within the document content.

Examples

```
<backm>
<index>
<gendtitle>
</index>
</backm>
```

Attributes

Toc=toc | notoc
Specifies whether the heading should appear in the table of contents. The default is for headings to appear in the table of contents; to a certain level (such as heading level 3). This replaces the older style=hidden attribute. Use this to prevent headings from appearing in the table of contents. It cannot be used to add a heading to the table of contents that would not normally appear. The levels of headings that appear in the table of contents are determined by the MaxToc attribute of the IBMIDDoc tag.

SPEC=AUTO | MAN
Specifies that the content of the element is generated. SPEC=AUTO is the default value.

| **LAYOUT=Default-Layout | OneCol | TwoCol | ThreeCol**
| Specifies the column-style for this portion of the book.

| **Default-Layout**
| The section uses the default layout for the document style.

| **OneCol**
| The entries format across the entire page.

| **TwoCol**
| The entries format in two columns.

| **ThreeCol**
| The entries format in three columns.

See “Common Element Attributes (large set)” on page 227.

Usage

See “Generating the index” on page 123.

Contexts

Children: GendTitle, RetKey, TitleBlk.

Parents: BackM.

Internet (internet e-mail address)

Purpose

The Internet element contains an Internet email address.

Examples

```
<address>
ATTN Dept 245
3605 Hwy 52 N
Rochester MN
<postalcode>55901-9986</postalcode>
<internet>fred@us.ibm.com</internet>
</address>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Contexts

Children: text (#pcdata).

Parents: Address.

IRef (index entry reference)

Purpose

The IRef element associates an element with an index entry by referring to an index entry defined elsewhere in the document, either in content or in an IdxDefs element.

Index entries specified in the information content are normally used to generate a traditional index. When index entries are specified in an IdxDefs element within Prolog or DProlog, they define index structures that can be used by reference from within the document content. Index entries within IdxDefs will not appear in a generated index unless specifically referred to.

Examples

```
<i1><idxterm>sauces</idxterm>
<i2 id="mayo"><idxterm>mayonnaise</idxterm></i2></i1>
 ...
<iref refids="mayo">
 ...
<iref refids="mayo">
```

Attributes

REFIDS=*index_entry_ids*
: Refers to one or more index entries (I1, I2, or I3) to be associated with the element that contains this IRef.

PRIMARY=PRIMARY
: Indicates that this entry is a primary entry for the term. The primary term is usually given some form of emphasis. There should be only one primary entry for each term.

See “Common Element Attributes (large set)” on page 227.

Usage

See “Cross referencing index entries” on page 118.

Contexts

Children: empty.

Parents: IdxDefs.

ISBN (document ISBN number)

Purpose

The ISBN element contains a document’s ISBN number.

Examples

```
<P>To better understand the intricacies of SGML, see
<CIT>
<BIBENTRY>
<DOCTITLE>
<TITLEBLK>
<TITLE>The SGML Handbook</TITLE>
```

```
</TITLEBLK>
<AUTHOR>
<NAME>Charles F. Goldfarb
</NAME>
</AUTHOR>
<PUBLISHER>
<CORPNAME>
 Oxford University Press
</CORPNAME>
<ADDRESS>
Walton St
Oxford OX2 6DP
</ADDRESS>
</PUBLISHER>
<PRTLOC>Printed and Bound in Great Britain
<ISBN>0-19-835737-9
<PUBID>+//ISBN 0-19-853737-9//DOCUMENT The SGML Handbook//EN
</BIBENTRY>
</CIT>
for more information.
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

For more information about the ISBN element, see "An example of using BibEntry and BibEntryDefs" on page 144.

Contexts

Children: text (#pcdata), Ph.

Parents: BibEntry, IBMBibEntry, IBMLibEntry, LibEntry.

I1 (primary index entry)

Purpose

The I1 element contains an index entry and related secondary or tertiary (third-level) entries. An index entry is associated with the element that directly contains it.

Index entries specified in the information content will be used to generate a traditional index. When index entries are specified in an IdxDefs element within Prolog or DProlog, they define index structures that can be used by reference from within the document content. Index entries within IndexDefs will not appear in a generated index unless specifically referred to.

Examples

```
<i1><idxterm>dessert sauces</idxterm></i1>
```

Attributes

| **SEEID=***i1_ids* | *i2_ids*
| Defines a SEE or SEE ALSO reference for cross-referencing. This points at one
| or more IDs on I1 or I2 index tags. Multiple IDs should be separated by

| blanks. Using the SEEID instead of SEETEXT ensures that there are
| corresponding index entries; because of the cross-reference. See "Defining See
| and See-also references" on page 121.

SEETEXT=*see_also_text*
Contains the text of a see or see also reference. For example, under an I1 entry of Poultry, you can use `SEETEXT="Chicken, Turkey, Quail, Duck, and Goose"`. When you specify the SEETEXT attribute, you must ensure that there are index entries for each word or phrase mentioned in the SEETEXT content. SEEID and SEETEXT can both be specified, if desired. SEEID takes precedence over SEETEXT. See "Defining See and See-also references" on page 121.

| **SORTKEY=***sortkey text*
| When specified, the SortKey= text is used to sort the entry, rather than the
| index entry text itself. This is not currently supported by Xyvision. See
| "Controlling the Index Sorting" on page 123.

PRIMARY=Primary
Indicates that this entry is a primary entry for the term. The primary term is usually given some form of emphasis. There should be only one primary entry for each term. Not supported for most of the IDWB transforms.

See "Common Element Attributes (large set)" on page 227.

Usage

See Chapter 11, "Indexing" on page 115.

Contexts

Children: I2, IdxTerm.

Parents: IdxDefs.

I2 (secondary index entry)

Purpose

The I2 element contains an index entry and any related tertiary (third-level) entries. An index entry is associated with the element that directly contains it. I2 outside the context of I1 must use the I1ID attribute to refer to an I1 element.

Index entries specified in the information content will be used to generate a traditional index. When index entries are specified in an IdxDefs element within Prolog or DProlog, they define index structures that can be used by reference from within the document content. Index entries within IndDefs will not appear in a generated index unless specifically referred to.

Examples

```
<i1><idxterm>dessert sauces</idxterm>
<i2><idxterm>butterscotch</idxterm></i2>
<i2><idxterm>hot fudge</idxterm>
<i3><idxterm>microwave method</idxterm></i3>
<i3><idxterm>stovetop method</idxterm></i3>
</i2>
<i2><idxterm>strawberry</idxterm></i2>
</i1>
```

Attributes

I1ID=*i1 id*
: Refers to the first level entry for this second level entry. I1ID is required when I2 occurs outside the context of an I1 element.

| **SEEID=***i1_ids* | *i2_ids*
: Defines a SEE or SEE ALSO reference for cross-referencing. This points at one or more IDs on I1 or I2 index tags. Multiple IDs should be separated by blanks. Using the SEEID instead of SEETEXT ensures that there are corresponding index entries; because of the cross-reference. See “Defining See and See-also references” on page 121.

SEETEXT=*see_also_text*
: Contains the text of a see or see also reference. For example, under an I2 entry of Poultry, you can use `SEETEXT="Chicken, Turkey, Quail, Duck, and Goose"`. When you specify the SEETEXT attribute, you must ensure that there are index entries for each word or phrase mentioned in the SEETEXT content. SEEID and SEETEXT can both be specified, if desired. SEEID takes precedence over SEETEXT. See “Defining See and See-also references” on page 121.

SORTKEY=*sortkey text*
: When specified, the SortKey= text is used to sort the entry, rather than the index entry text itself. This is not currently supported by Xyvision. See “Controlling the Index Sorting” on page 123.

PRIMARY=Primary
: Indicates that this entry is a primary entry for the term. The primary term is usually given some form of emphasis. There should be only one primary entry for each term. Not supported for most of the IDWB transforms.

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 11, “Indexing” on page 115.

Contexts

Children: I3, IdxTerm.

Parents: I1, IdxDefs.

I3 (tertiary index entry)

Purpose

The I3 element contains a third-level index entry. An index entry is associated with the element that directly contains it. I3 outside the context of I2 must use the I2ID attribute to refer to an I2 element.

Index entries specified in the information content will be used to generate a traditional index. When index entries are specified in an IdxDefs element within Prolog or DProlog, they define index structures that can be used by reference from within the document content. Index entries within IndexDefinition will not appear in a generated index unless specifically referred to.

Examples

```
<i1><idxterm>dessert sauces</idxterm>
<i2><idxterm>butterscotch</idxterm></i2>
<i2><idxterm>hot fudge</idxterm>
<i3><idxterm>microwave method</idxterm></i3>
<i3><idxterm>stovetop method</idxterm></i3>
</i2>
<i2><idxterm>strawberry</idxterm></i2>
</i1>
```

Attributes

I2ID=*i2 id*
: Refers to the first level entry for this second level entry. I2ID is required when I3 occurs outside the context of an I2 element.

SORTKEY=*sortkey text*
: When specified, the SORTKEY text is used to sort the entry, rather than the index entry text itself.

PRIMARY=Primary
: Indicates that this entry is a primary entry for the term. The primary term is usually given some form of emphasis. There should be only one primary entry for each term.

Usage

See Chapter 11, “Indexing” on page 115.

Contexts

Children: IdxTerm.

Parents: I2, IdxDefs.

Kwd (syntax keyword)

Purpose

Use Kwd to define keywords within a syntax definition. Keywords are literal values that must be specified exactly as shown in the diagram.

Examples

```
<syntax>
<group>
<kwd>FORM</kwd>
<kwd optreq="opt">PROC</kwd>
</group>
</syntax>
```

Attributes

ABBREVS=*abbreviations*
: Lists the valid blank delimited abbreviations for the keyword.

OPTREQ=REQ | OPT | DEF
: Indicates whether or not the keyword is optional. REQ (required) is the default.

LINKEND=*id*
Contains an ID reference that enables a link to an arbitrary location in the document.

See “Common Element Attributes (large set)” on page 227.

Usage

See “The KWD (keyword) element” on page 152.

Contexts

Children: text (#pcdata).

Parents: Group, SynPh.

L (explicit link)

Purpose

The L element links a phrase to any place in a document, another document, or a non-text object, such as a multimedia presentation.

L can point to either another element in the same document, or it can point to a NameLoc element, through which it can link to almost anything. What you can link to via NameLoc is limited only by the online presentation system you use.

Note: IBMIDDoc neither defines nor limits the types of things you can link to from a document. The linking you can do is determined by the online presentation system you are using.

As a rule, it is worth using NameLoc when something will be linked to more than twice within the same document, because the indirection provided by NameLoc makes maintaining those links easier.

Examples

```
<d id="xrefhyl">
<dprolog><titleblk>
<title>All about linking</title>
</titleblk></dprolog>
<dbody>
<p>Hypertext links (we'll just call them links from
now on) connect elements in one part of an online
document to elements in another part of the same document
or a separate online document. </p>
 ...
<p>Sometimes you need to <l linkend="xrefhyl">link</l> to
other topics.</p>
```

Attributes

LINKEND=*element_id*
The ID value of the element being linked to or a NameLoc element that ultimately locates the object or objects being linked to.

SPEC=AUTO
Currently does not work.

CLASS
You can use this to affect whether that link will replace the content of a frame; or whether the link will launch a new browser window.

NewWindow
This opens the link in a new, unnamed window. This is the same as the HTML coding: `target="_blank"`

FullWindow
This opens the link into the full, original window, cancelling all frames. This is the same as the HTML coding: `target="_top"`

SameWindow
This opens the link into the same window. This is the same as the HTML coding: `target="_self"`

See "Common Element Attributes (large set)" on page 227.

Usage

See Chapter 12, "All about linking" on page 129.

Contexts

Children: text (#pcdata), Address, APL, Bin, Char, Cit, Date, Dec, Formula, Hex, L, MD, MV, Num, Oct, Ph, PK, PV, Q, RefKey, SynPh, Term, TM, XPh, XRef.

Parents: Address, AnnotBody, Attention, Bridge, Cap, Caution, CGraphic, CI, CLE, CompCmt, Cond, Copyr, Danger, DBody, Defn, DefnHd, Desc, DIntro, DSum, EdNotices, entry, Fig, FigSeg, Fn, L, LEDesc, LEDI, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, ModName, MsgItem, MsgText, NItem, NoteBody, Notices, P, PBlk, Ph, ProcEntry, ProcExit, ProcIntro, Q, Safety, Screen, Sem, STitle, SubTitle, SynNote, Term, TermHd, TextAlt, Title, Warning, Xmp, XPh.

LDescs (link descriptions)

Purpose

Use LDescs to contain descriptions of links that need to be referenced from more than one place the document.

Examples

```
<PROLOG>
 <IBMBIBENTRY ID="BOOKMSG">
  <DOCTITLE><TITLEBLK><TITLE>IBM BOOKMASTER USER'S GUIDE </TITLE>
   </TITLEBLK>
  </DOCTITLE>
  <IBMDOCNUM>SC34-5107</IBMDOCNUM>
   <DESC>DESCRIBE HOW TO USE BOOKMASTER</DESC>
 </IBMBIBENTRY>
 <LDESCS>
  <NAMELOC ID="ABC1" OBJTYPE="BOOK">
   <NMLIST></NMLIST>
  </NAMELOC>
  <NAMELOC ID="ABC0" OBJTYPE="HEAD">
   <NMLIST DOCNAME="BOOKUG">SYMBS</NMLIST>
  </NAMELOC>
 </LDESCS>
 <BIBENTRYDEFS>
  <IBMBIBENTRY ID="BOOKUG">
   <DOCTITLE><TITLEBLK><TITLE>TITLE</TITLE></TITLEBLK>
   </DOCTITLE>
   <IBMDOCNUM>SN23-0059</IBMDOCNUM>
    <DESC>BOOK DESCRIPTION</DESC>
  </IBMBIBENTRY>
 </BIBENTRYDEFS>
</PROLOG>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Using LDescs and Nameloc” on page 95.

Contexts

Children: AreaDef, nameloc, notloc.

Parents: DProlog, Prolog, SpecDProlog.

LE (language element)

Purpose

The LE element contains the description of a computer language element, such as a command, within the context of a language element reference section. Use LE and LERS to create reference information for computer languages such as command sets, programming languages, and the like. The presentation of language elements is as headed sections.

Examples

```
<LERS>
 <LE>
  <LEN>aname
  </LEN>
<LEDI CLASS="PURPOSE">
 <P>Use this command to request help information on the APPC NameServer
   facility.
<LEDI CLASS="FORMAT">
 <SYNTAX>
  <GROUP>
   <KWD OPTREQ="REQ">aname
   <KWD OPTREQ="REQ"> -h
  </GROUP>
 </SYNTAX>
<LEDI CLASS="PARMS">
 <PARML STYLE="BKM:(BREAK='NONE' TSIZE='&bigt.')">
  <PARM>
   <TERM>-h</TERM>
   <DEFN>An explicit request for help information on the ANAME command.
   </DEFN>
 </PARML>
<LEDI CLASS="EXAMPLES">
 <P>This example requests general help on the ANAME command.
  <XMP>
   aname -h
  </XMP>
 </LE>
</LERS>
```

Attributes

Toc=toc | notoc
Specifies whether the heading should appear in the table of contents. The default is for headings to appear in the table of contents; to a certain level (such as heading level 3). This replaces the older style=hidden attribute. Use this to prevent headings from appearing in the table of contents. It cannot be used to add a heading to the table of contents that would not normally appear. The levels of headings that appear in the table of contents are determined by the MaxToc attribute of the IBMIDDoc tag.

See "Common Element Attributes (large set)" on page 227.

Usage

See Chapter 16, "Developing Programming Language Reference Materials" on page 165.

Contexts

Children: LEDesc, LEDI, LEN, RetKey.

Parents: LERS.

LeDesc (language element description)

LeDesc can contain the description of the command. This is usually a short abstract.

Purpose

The LEDI element's Purpose class should be used to contain the main purpose of the language element.

Examples

```
<le id="LeDesc">
<len>LeDesc (language element description)</len>
<ledesc>LeDesc can contain the description of the
command. This is usually a short abstract. </ledesc>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See Chapter 16, "Developing Programming Language Reference Materials" on page 165.

Contexts

Children: text (#pcdata), DL, Fig, L, MMObj, Note, NoteList, OL, P, PBlk, Ph, Table, Term, TM, UL.

Parents: LE.

LEDI (language element description item)

Purpose

The LEDI element contains a description of one aspect of a language element within a LERS section. Each language element can have several LEDI elements.

For each LEDI, the title text can be defined separately on a LERSDEF element. Each LEDI class can have a different generated title or no title. The default presentation style for language element descriptions is as normal divisions.

Examples

```
<LERS>
 <LE>
  <LEN>aname
  </LEN>
<LEDI CLASS="PURPOSE">
 <P>Use this command to request help information on the APPC NameServer
    facility.
<LEDI CLASS="FORMAT">
 <SYNTAX>
  <GROUP>
   <KWD OPTREQ="REQ">aname
   <KWD OPTREQ="REQ"> -h
  </GROUP>
 </SYNTAX>
<LEDI CLASS="PARMS">
 <PARML STYLE="BKM:(BREAK='NONE' TSIZE='&bigt.')">
  <PARM>
   <TERM>-h</TERM>
   <DEFN>An explicit request for help information on the ANAME command.
   </DEFN>
 </PARML>
<LEDI CLASS="EXAMPLES">
 <P>This example requests general help on the ANAME command.
  <XMP>
   aname -h
  </XMP>
 </LE>
</LERS>
```

Attributes

Class Values
Indicates the class of the LEDI. The class values define what type of information each LEDI contains:

AUTH
Authorization level; for example, user or superuser.

COMMENTS
Comments about the language element.

CONTEXT
The context in which the language element is used.

DEFAULTS
The defaults for the language element.

ERRCOND
Error conditions.

EXAMPLES
Examples of using the language element.

FLAGS
Control flags.

FORMAT
The format or syntax of the language element.

INTREP
Internal representation, such as control blocks or data structures.

MESSAGES
Any messages related to the language element.

OTHER
An open category.

PARMS
Parameters.

PROCESS
Processing related to the language element.

PURPOSE
The purpose of the language element.

RESTRICT
Restrictions on the use of the language element.

RESULTS
The results of using the language element.

RETCODES
Return codes.

SYSENV
The system environments to which the language element applies.

USAGE
How to use the language element.

VERSION
Any version information for the language element.

See “Using LDescs and Nameloc” on page 95.

Usage

See Chapter 16, “Developing Programming Language Reference Materials” on page 165.

Contexts

Children: Annot, AsmList, Attention, BibList, Bridge, Caution, CGraphic, D, Danger, DBlk, DL, Fig, FnList, GL, L, LERS, Lines, LitData, LQ, MarkList, MkNote, MMObj, ModInfo, MsgList, Note, NoteList, OL, P, ParmL, PartAsm, PBlk, Proc, Screen, Syntax, Table, UL, Xmp.

Parents: LE.

Legend

Purpose

The Legend element contains an explanation of any special notations used in the document, such as within graphics, tables, figures, or other specialized information.

Examples

```
<FRONTM>
 <LEGEND>
  <SPECDPROLOG>
   <GENDTITLE>
  </SPECDPROLOG>
  <DBODY>
   <P>The following symbols have special meaning in this document:
   ⋮
  </DBODY>
 </LEGEND>
</FRONTM>
```

Attributes

| **LAYOUT=Document-Layout | OneCol | TwoCol | OffsetCol**
| Specifies the column-style for this portion of the book.

| **Document-Layout**
| The section uses the default layout for the document style.

| **OffsetCol**
| Formats the text using an indented, one-column layout. The headings
| format across the page, with the offset text, or indented from the
| margin.

| **OneCol**
| The text formats across the entire page.

| **TwoCol**
| The text formats in two columns.

Toc=toc | notoc
Specifies whether the heading should appear in the table of contents. The default is for headings to appear in the table of contents; to a certain level (such as heading level 3). This replaces the older style=hidden attribute. Use this to prevent headings from appearing in the table of contents. It cannot be used to add a heading to the table of contents that would not normally appear. The levels of headings that appear in the table of contents are determined by the MaxToc attribute of the IBMIDDoc tag.

See “Common Element Attributes (large set)” on page 227.

Usage

See “Special sections” on page 100.

Contexts

Children: DBody, DIntro, DSum, SpecDProlog.

Parents: FrontM.

LEN (language element name)

Purpose

Use LEN for the name of the language element, for example, the command name.

Examples

```
<le id="len">
<len>LEN (language element name)</len>
<ledi class="PURPOSE">
<p>Use LEN for the name of the language element, for
example, the command name.</p>
</ledi>
<ledi class="EXAMPLES">
<xmp></xmp>
</ledi>
<ledi class="PARMS">
<p conloc="commattr">
</ledi>
<ledi class="USAGE">
<p>See <xref refid="langref">.</p>
</ledi>
<ledi class="CONTEXT"><pblk conloc="context_len">
</ledi>
</le>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 16, “Developing Programming Language Reference Materials” on page 165.

Contexts

Children: text (#pcdata), Ph, TM.

Parents: LE.

LERS (language element reference section)

Purpose

Use LERS to contain reference information for computer languages such as programming languages, command sets, and the like. Use the more generic ModInfo elements for other sorts of modular information. A LERS section contains one or more language element descriptions (LE). The default presentation style for LERS is as normal divisions.

Examples

```
<le id="len">
<len>LEN (language element name)</len>
<ledi class="PURPOSE">
<p>Use LEN for the name of the language element, for
example, the command name.</p>
</ledi>
<ledi class="EXAMPLES">
<xmp></xmp>
```

```
</ledi>
<ledi class="PARMS">
<p conloc="commattr">
</ledi>
<ledi class="USAGE">
<p>See <xref refid="langref">.</p>
</ledi>
<ledi class="CONTEXT"><pblk conloc="context_len">
</ledi>
</le>
```

Attributes

DEF=*definition-name*
: Specifies the definition to be used. This points to a DefName attribute on a corresponding DEF tag. The settings defined on that DEF tag are used as defaults for this tag. See Chapter 9, "Using definition tags" on page 105 for more information.

CLASS=*classname*
: **Deprecated — use DEF attribute.** References the CLASSNAME attribute on a LersDef element.

COMPLANG=*computer-language*
: Specifies the computer language described in this LERS section.

LAYOUT=Document-Layout | OneCol | TwoCol | OffsetCol
: Specifies the column-style for this portion of the book.

  Document-Layout
  : The section uses the default layout for the document style.

  OffsetCol
  : Formats the text using an indented, one-column layout. The headings format across the page, with the offset text, or indented from the margin.

  OneCol
  : The text formats across the entire page.

  TwoCol
  : The text formats in two columns.

RETKEY=None | First
: Use the RetKey attribute to enable or disable automatic running headings for this tag.

  NONE
  : indicates that nothing is to be used.

  FIRST
  : indicates that the first non-blank item on the page is to be used.

  The values NODUP and FIRSTLAST are not supported at this time.

  If you code any explicit RetKey elements, they are honored and will appear. If you nest elements that can generate a running heading (for example, a MsgList inside Lers), only the outer active generated heading is used. That is, if you specified automated RetKey generation for LERS and MSGLIST, a Msgno inside Lers will not be used in the RetKey area. But if you had an explicit RetKey inside the Msg, then the RetKey is honored as an explicit override.

classname=*title_text*
For each LEDI class, you can define the generated title for the LEDI elements.

AUTH
Authorization level; for example, user or superuser.

COMMENTS
Comments about the language element.

CONTEXT
The context in which the language element is used.

DEFAULTS
The defaults for the language element.

ERRCOND
Error conditions.

EXAMPLES
Examples of using the language element.

FLAGS
Control flags.

FORMAT
The format or syntax of the language element.

INTREP
Internal representation, such as control blocks or data structures.

MESSAGES
Any messages related to the language element.

OTHER
An open category; define your own title.

PARMS
Parameters.

PROCESS
Processing related to the language element.

PURPOSE
The purpose of the language element.

RESTRICT
Restrictions on the use of the language element.

RESULTS
The results of using the language element.

RETCODES
Return codes.

SYSENV
The system environments to which the language element applies.

USAGE
How to use the language element.

VERSION
Any version information for the language element.

| **SEP= PAGE | NORMAL | LHPAGE | RHPAGE**
| allows you to specify how you want the language elements separated, where:

| **PAGE** Starts the language element on the next page.

NORMAL
Specifies normal heading separation — usually white space.

LHPAGE
Starts the language element on the next left-hand page (even page).

RHPAGE
Starts the language element on the next right-hand page (odd page).

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 16, “Developing Programming Language Reference Materials” on page 165.

Contexts

Children: LE, RetKey.

Parents: Appendix, Body, DBody, LEDI, ModItem, MsgItem.

LERSDef (LERS property definition)

Purpose

Use LERSDef to define the titles to be generated for different classes of LEDI elements, and to define values for common properties.

Examples

```
<lersdef defname="taglers" comments="Usage" context="Contexts"
defaults="Style Values" examples="Examples" format="Syntax"
other="SGML Markup" parms="Attributes" results="More Information"
usage="Usage">
</lersdef>
```

Attributes

DEFNAME=*definition-name*
The DefName attribute identifies a definition element. DefName values must be unique within a single document. A element that has a DEF attribute can point a corresponding tag with a DefName attribute. DefName values can be up to 64 characters long. DefName values must start with an alphabetic character and can contain letters, numbers, dashes (-), or periods (.).

Definition tags can be specified without a DefName attribute in the document’s prolog. This allows you to change the initial settings of attributes for the entire document when the DEF tag is in the document’s prolog. In a division’s prolog, it changes the initial settings for that division.

RETKEY=None | First
Use the RetKey attribute to enable or disable automatic running headings for this tag.

NONE
indicates that nothing is to be used.

FIRST
indicates that the first non-blank item on the page is to be used.

The values NODUP and FIRSTLAST are not supported at this time.

If you code any explicit RetKey elements, they are honored and will appear. If you nest elements that can generate a running heading (for example, a MsgList inside Lers), only the outer active generated heading is used. That is, if you specified automated RetKey generation for LERS and MSGLIST, a Msgno inside Lers will not be used in the RetKey area. But if you had an explicit RetKey inside the Msg, then the RetKey is honored as an explicit override.

CLASSNAME = *classname*
: **Depreciated — use DEFNAME attribute.** Defines the name of the class. This name is the name referenced by the CLASS attribute on the Lers element.

COMPLANG=*computer-language*
: Specifies the computer language described in this LERS section.

*classname***=***title_text*
: For each LEDI class, you can define the generated title for the LEDI elements.

AUTH
: Authorization level; for example, user or superuser.

COMMENTS
: Comments about the language element.

CONTEXT
: The context in which the language element is used.

DEFAULTS
: The defaults for the language element.

ERRCOND
: Error conditions.

EXAMPLES
: Examples of using the language element.

FLAGS
: Control flags.

FORMAT
: The format or syntax of the language element.

INTREP
: Internal representation, such as control blocks or data structures.

MESSAGES
: Any messages related to the language element.

OTHER
: An open category; define your own title.

PARMS
: Parameters.

PROCESS
: Processing related to the language element.

PURPOSE
: The purpose of the language element.

RESTRICT
: Restrictions on the use of the language element.

RESULTS
The results of using the language element.

RETCODES
Return codes.

SYSENV
The system environments to which the language element applies.

USAGE
How to use the language element.

VERSION
Any version information for the language element.

SEP= PAGE | NORMAL | LHPAGE | RHPAGE
allows you to specify how you want the language elements separated, where:

PAGE Starts the language element on the next page.

NORMAL
Specifies normal heading separation — usually white space.

LHPAGE
Starts the language element on the next left-hand page (even page).

RHPAGE
Starts the language element on the next right-hand page (odd page).

See “Common Element Attributes (large set)” on page 227.

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 16, “Developing Programming Language Reference Materials” on page 165.

Contexts

Children: Desc.

Parents: PropDefs, PropGroup.

LI (list item)

Purpose

The LI element contains a single list item within a list. List items can be grouped together with LiBlk..

Examples

```
<ul>
<li>This is an item in an unordered list. To separate
it from other items in the list, the formatter puts
a bullet beside it.</li>
<li>The paragraph that is contained in the LI element
is part of the list item which contains it. <p>This
is the contained paragraph.</p></li>
<li>This is a separate list item in our unordered
list.</li>
</ul>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Unordered lists” on page 29.

Contexts

Children: text (#pcdata), Address, Annot, APL, Attention, Bin, Bridge, Caution, CGraphic, Char, Cit, Danger, Date, Dec, DL, Fig, Formula, GL, Hex, L, Lines, LitData, LQ, MD, MkNote, MMObj, ModInfo, MV, Note, NoteList, Num, Oct, OL, P, ParmL, PBlk, Ph, PK, PV, Q, RefKey, Screen, SynPh, Syntax, Table, Term, TM, UL, Xmp, XPh, XRef.

Parents: LIBlk, NoteList, OL, StepNotes, UL.

LibEntry (document library definition)

Purpose

Use LibEntry to define the bibliographic information about a library or other collection of documents. You can use the CLASS attribute and the ClassDef element to define different classes of LibEntry to correspond to different classes of collection. For example, you may have product libraries that are themselves collected into larger libraries of libraries. You could define a class of "CollectionKit" for libraries of libraries and then use LibEntry elements to define the contents of a given collection kit.

LibEntry elements can also be specified in a BibEntryDefs section or object container and used by reference from within a document, for example, from Cit elements. When LibEntry elements are specified within Cit, the LibEntry elements are collected for use in a generated bibliography.

Contained BibEntry elements and LibEntry elements are normally used within re-used information in order to ensure that the re-used information is completely self-contained. In other words, these elements should be used within the scope of the information that is being re-used. For example, if a Division element is re-used, the BibEntry and LibEntry elements should be contained within that same division’s DProlog element. This allows these elements to be completely contained, and thus completely re-usable, within the division that is being re-used.

It is intended that the public identifier of the library entity be used by the presentation system to locate the actual system object, but specific presentation systems may define application-specific data to be specified as the system identifier of the library entity if they do not support the use of public identifiers. The public identifier can be included in the LibEntry itself as a way of keeping a library’s formal public identifier definition with the rest of its bibliographic information. This could allow, for example, the automatic generation of entity declarations for libraries described by LibEntry elements.

Examples

```
<bibentrydefs>
<ibmlibentry>
<library><titleblk><title>BS/300</title></titleblk>
</library>
<ibmbofnum>SBOF-1234-0</ibmbofnum>
<containeddocs bibids="booka bookb"></ibmlibentry>
<ibmbibentry id="booka"><doctitle><titleblk><title>
BS/300 Guide</title></titleblk></doctitle></ibmbibentry>
<ibmbibentry id="bookb"><doctitle><titleblk><title>
BS/300 Reference</title></titleblk></doctitle></ibmbibentry>
<libentry>
<library><titleblk><title>Back'n'Recovery</title>
</titleblk></library>
</libentry>
</bibentrydefs>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See "Defining library entries" on page 143.

Contexts

Children: BOFNum, ContainedDocs, Desc, ISBN, Library, OrderNum, ProdName, PrtLoc, PublicId, Publisher.

Parents: BibEntryDefs, BibList, Cit.

LIBlk (list item block)

Purpose

The LIBlk element groups items within a list. The reasons for the grouping are determined by the author. The grouping may be logical, and can be indicated by including an optional title. LiBlk can also be used define blocks of list items to be connected with a Bridge element.

If you want to have all the text in the LIBLK to be on the same page, use style="xpp:(keep)". Be cautious when using this feature. If the text does not fit on a page, remove the style="xpp:(keep)" or the pages will not format correctly.

Migration Note

The use of LIBlk, its title, or the use of Bridge, replaces the List Part (LP) element from BookMaster.

Examples

```
<ol>
<liblk>
<li>1 GIG SCSI-2 Hard Disk</li>
<li>32 MB RAM</li>
<li>128-Bit 8MB VRAM Video</li>
<li>21-Inch Monitor</li>
</liblk>
<liblk>
```

```
<li>Great Word Processor</li>
<li>Best Multimedia App</li>
<li>Voice Mail</li>
</liblk>
</ol>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Grouping list items” on page 37.

Contexts

Children: Bridge, LI, Title.

Parents: NoteList, OL, StepNotes, UL.

Library

Purpose

The Library element contains the name of a library.

Examples

```
See these books for a good read and then a weird read: <cit>
<bibentry><doctitle><titleblk><title>Tom Sawyer</title>
</titleblk></doctitle></bibentry></cit> and <cit>
<ibmbibentry><doctitle>
<library><titleblk><title>System/36</title></titleblk>
</library>
<titleblk><title>Concepts and Programmer's Guide</title>
</titleblk></doctitle></ibmbibentry></cit>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

For more information about the Library element, see “An example of using BibEntry and BibEntryDefs” on page 144.

Contexts

Children: TitleBlk.

Parents: DocTitle, IBMLibEntry, LibEntry.

Lines (text with line boundaries)

Purpose

The Lines element contains text for which the input line (record) boundaries are significant and must be preserved or indicated when presented.

Examples

```
<LINES>
a partridge in a pear tree
two turtledoves
three French hens
four calling birds
five golden rings
six geese a-laying
seven swans a-swimming
eight maids a-milking
nine ladies dancing
ten lords a-leaping
eleven pipers piping
twelve drummers drumming
</LINES>
```

Attributes

NOTATION=LINESPEC
Specifies that LINESPEC is the default value for the NOTATION attribute.

OBJ=*entity_name*
The name of the external data entity that contains the line-specific data. When OBJ is specified, it is an error to specify any data or the Lines end tag.

See "Common Element Attributes (large set)" on page 227.

Usage

See "Just plain lines" on page 53.

Contexts

Children: text (#pcdata), Address, APL, Bin, Char, Cit, Date, Dec, Formula, Hex, L, MD, MV, Num, Oct, Ph, PK, PV, Q, RefKey, SynPh, Term, TM, XPh, XRef.

Parents: AnnotBody, Attention, BackCover, Bridge, Caution, Danger, DBody, Defn, DIntro, DSum, entry, Fig, FigSeg, Fn, FrontCover, LEDI, LI, LQ, MkNote, ModDesc, ModItem, MsgItem, MsgText, NoteBody, P, PBlk, ProcIntro, SynNote, Warning.

Litdata (literal data)

Purpose

The Litdata element contains or refers to literal data in a specific notation. Use Litdata to contain or refer to data that is not to be parsed by the SGML parser and that may need special processing in order to properly present the data, such as character translation or use of special code pages.

When the OBJ attribute is used to refer to an external entity, the Litdata end tag cannot be specified. Do not specify a notation when referring to an external entity because the entity will have a notation defined on its entity declaration.

The typical use of Litdata is to contain or refer to samples of programming code.

The only SGML markup recognized within Litdata is the end tag open delimiter (</). When literal data is included inline in the document, the end tag open delimiter ends the Litdata element, regardless of the context.

Examples

```
<!ENTITY  testprg SYSTEM "testprg.c" ndata c>
   ⋮
<FIG>
<CAP>A basic C++ program.
<LITDATA OBJ="TESTPRG">
</FIG>
```

Attributes

NOTATION=*notation_name*
The name of the notation that the data is in. The notations supported are defined by the specific implementation of IBMIDDoc you are using, but typical notations include:

LINESPEC
Specifies that the line ends are respected. This is the default.

C
Cpp
C and C++ program source.

The processing application uses the notation name to determine what special processing is needed to present the literal data. For example, in the case of C program code, the square bracket and curly bracket characters may need special treatment.

OBJ=*entity_name*
Specifies the name of an external data entity that contains the literal data. When OBJ is specified, the Litdata end tag cannot be specified.

CDATA
Is the literal data.

Usage

See “Literal text data” on page 54.

Contexts

Children: text (cdata).

Parents: AnnotBody, Attention, BackCover, Bridge, Caution, CGraphic, Danger, DBody, Defn, DIntro, DSum, entry, Fig, FigSeg, Fn, FrontCover, LEDI, LI, LQ, MkNote, ModDesc, ModItem, MsgItem, NoteBody, P, PBlk, ProcIntro, Screen, SynNote, Warning, Xmp.

LQ (excerpt quotation)

Purpose

Use the LQ element to contain material excerpted from another source that can stand alone without a defining context. This is usually the case when the quotation is of considerable length. The LQ may be of considerable length and can contain almost any element, including divisions.

Examples

```
<lq>The only thing we have to fear is fear itself.
</lq>
```

LQ (excerpt quotation)

Attributes

BibId=*bibentry_id*
: The ID of the BibEntry element that defines the source of the quotation. You must either specify BIBID or include a BibEntry element within the LQ element.

See “Common Element Attributes (large set)” on page 227.

Usage

See “Quotes and excerpts” on page 47.

Contexts

Children: text (#pcdata), Address, Annot, APL, Attention, Bin, Bridge, Caution, CGraphic, Char, Cit, Danger, Date, Dec, DL, Fig, Formula, GL, Hex, L, Lines, LitData, MD, MkNote, MMObj, ModInfo, MV, Note, NoteList, Num, Oct, OL, P, ParmL, PBlk, Ph, PK, PV, Q, RefKey, Screen, SynPh, Syntax, Table, Term, TM, UL, Xmp, XPh, XRef.

Parents: AnnotBody, Attention, Bridge, Caution, Danger, DBody, Defn, DIntro, DSum, entry, Fig, FigSeg, Fn, LEDI, LI, ModDesc, ModItem, MsgItem, NoteBody, P, PBlk, ProcIntro, SynNote, Warning.

Maintainer (reader comment)

Purpose

The content of the Maintainer element is used for tracking control information. It is also used in generating a Reader Comment Form. If this information is not included, the Reader Comment Form cannot be generated by the output processor.

Usage

See “Using reader’s comment form (RCF)” on page 103.

Examples

```
<maintainer>
<corp>
<corpname>IBM Corporation</corpname>
<address>ATTN: Dept 542
3605 HWY 52 N
Rochester, MN
<postalcode>55901-9986</postalcode></address>
</corp>
</maintainer>
```

Contexts

Children: Corp, Person.

Parents: DProlog, Prolog, SpecDProlog.

Mark (marked note definition)

Purpose

Use Mark to define a collection of marked notes. To use marked notes, you must define at least one collection in a document. All marked notes must be associated with a collection.

You can define different collections of notes to correspond to different releases of a document or to different types of notes. Different collections may use the same classes and actions. For example, if you use marked notes to track changes and generate a summary of changes, you can define different collections for different releases or drafts of a document. Or if you are tracking changes from different sources, you can define a collection for each different source.

Examples

```
<mark id="mkv5r4" ident="use">
<desc>V5R4 marked changes</desc>
</mark>
```

Attributes

ID=*collection_id*
Defines the mark collection ID.

IDENT=USE | IGNORE
Determines whether to process the specified collection of marked notes as follows:

USE
Process the collection of marked notes.

IGNORE
Ignore the collection of marked notes.

Conloc
The CONLOC attribute specifies that the content of another element of the same type is to be used as the content of the referencing element. This enables reuse of information. When the CONLOC attribute is specified, you cannot specify the element's end tag. The result of using CONLOC is exactly the same as if the element being referred to had occurred at that point in the document. See "Reusing elements from an object library" on page 191.

Attributes on the element are now passed to the element with the CONLOC; this is a feature that began with IDWB release 3.4, patch IDWXF036.

Class
The Class attribute associates an element class with an element. This attribute must contain an SGML name that has been defined as a class name in a ClassDef element. This attribute only applies to elements specified on the ClassDef's ELETYPE attribute. Element classes are defined with ClassDef elements within PropDefs. See Chapter 20, "Property and Class Definitions" on page 201 for more information.

Rev
The REV attribute references the Rev element which describes the last revision level of the information. See "Using Revisions" on page 109.

Status
ignored by processes

InfoMast
A fixed attribute used to classify the element.

Usage

See "Creating Collections of Marked Notes" on page 111.

Contexts

Children: Desc.

Parents: RevDefs.

MarkList (marked note list)

Purpose

Use MarkList to include a list of marked notes. Only notes that are members of the specified collections and that have the specified class and action values are included in the list.

Examples

```
<marklist mkids="mkv5r4" classes="msg abend" actions="change del rep"
display="item action location desc" classhd="Msg"
actionhd="Reason" lochd="Loc" deschd="Desc">
```

Attributes

ID=*marklist_id*
Specifies the ID of this element.

DISPLAY=NAMES

CLASSES=*class_name*
Defines the mark class or classes to include in this list. Only notes with a specified class will be included.

ACTIONS=*action_name*
Defines the action or actions to include in this list. Only notes with a specified action will be included.

SPEC=AUTO | MAN
Specifies that the content of the element is generated.

Future Enhancement
At this time, MAN is not supported.

MKIDS=*mark_id*
Defines which collections to search for notes to present.

DISPLAY=CLASS | ACTION | LOC | ITEM | DESC
Specifies the type of information to display in the generated list. You can choose one or more items from the group, but you can choose each item only once. The items in the list are displayed in the order they are specified on the Display attribute.

CLASS
Specifies that the CLASS attribute values for the marked notes are to be included.

ACTION
Specifies that the ACTION attribute values for the marked notes are to be displayed.

ITEM
Specifies that the ITEM attribute values for the marked notes are to be displayed.

LOC
Specifies that the locations of the marked notes, either page numbers or online equivalents, are to be displayed.

DESC
Specifies that the contents of the marked notes are to be displayed.

CLASSHd=*column_heading*
ACTIONHd=*column_heading*
ITEMHd=*column_heading*
LOCHd=*column_heading*
DESCHd=*column_heading*
Defines the heading text associated with each type of information in the generated list. These values override the default headings defined in the document style.

Toc=toc | notoc
Specifies whether the heading should appear in the table of contents. The default is for headings to appear in the table of contents; to a certain level (such as heading level 3). This replaces the older style=hidden attribute. Use this to prevent headings from appearing in the table of contents. It cannot be used to add a heading to the table of contents that would not normally appear. The levels of headings that appear in the table of contents are determined by the MaxToc attribute of the IBMIDDoc tag.

See “Common Element Attributes (large set)” on page 227.

Usage

See “Creating Collections of Marked Notes” on page 111.

Contexts

Children: empty.

Parents: DBody, DIntro, DSum, LEDI, MsgItem, PBlk, ProcIntro.

MasterIndex (master index)

Purpose

The Master Index is a way to combine the indexes of several documents. Rather than having to look in the index of several documents, the user can look in the master index for the correct document and page number where the index entry is located.

Examples

```
<backm>
<masterindex>
<specdprolog><gendtitle></specdprolog>
<masterindexobj obj="gsugidx">
<masterindexobj obj="planidx">
<masterindexobj obj="instidx">
</masterindex></backm>
```

Attributes

| **LAYOUT=Default-Layout | OneCol | TwoCol | ThreeCol**
| Specifies the column-style for this portion of the book.

| **Default-Layout**
| The section uses the default layout for the document style.

| **OneCol**
| The entries format across the entire page.

| **TwoCol**
| The entries format in two columns.

| **ThreeCol**
| The entries format in three columns.

See “Common Element Attributes (large set)” on page 227.

Contexts

Children: DIntro, DSum, MasterIndexObj, SpecDProlog.

Parents: BackM.

MasterIndexInfo (master index information)

Purpose

MasterIndexInfo is used in conjunction with MasterIndexPrefix. Use these tags when preparing to merge the indexes of several documents into a master index. These tags should be used in the content of each document containing an index you wish to have merged into a master index.

Examples

```
<ibmbibentry><doctitle>
<library><titleblk>
<title>ID Workbench</title>
</titleblk></library>
<titleblk>
<title>Getting Started and User's Guide</title>
</titleblk></doctitle>
<externalfilename>idfgsmst</externalfilename>
</ibmbibentry>
<masterindexinfo>
<masterindexprefix>GSUG</masterindexprefix>
</masterindexinfo>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Contexts

Children: MasterIndexPrefix.

Parents: DProlog, Prolog, SpecDProlog.

MasterIndexObj (master index object)

Purpose

MasterIndexObj is used with the MasterIndex tag. The MasterIndexObj tag provides a reference to the index of a document to be included in the Master Index. This is a separate document in itself. It is separate from the documents with indexes you are merging.

Examples

```
<!ENTITY guide SYSTEM "rweguide.mdx" ndata mindex>
<!ENTITY ref SYSTEM "rweref.mdx" ndata mindex>
 ...
<masterindex>
<specdprolog><gendtitle></specdprolog>
<masterindexobj obj="guide">
<masterindexobj obj="ref">
</masterindex>
```

Attributes

OBJ
Specifies the name of the master index entity.

Class
The Class attribute associates an element class with an element. This attribute must contain an SGML name that has been defined as a class name in a ClassDef element. This attribute only applies to elements specified on the ClassDef's ELETYPE attribute. Element classes are defined with ClassDef elements within PropDefs. See Chapter 20, "Property and Class Definitions" on page 201 for more information.

ID=*identifier*
The ID attribute identifies an element within an SGML document. IDs must be unique within a single document. Any element that has an ID can be cross-referenced or linked to. IDs can be up to 64 characters long. IDs must start with an alphabetic character and can contain letters, numbers, dashes (-), or periods (.).

Props=*properties*
The Props attribute is used to specify the condition under which the information contained within the element appears. See "Property-Based Retrieval" on page 195.

PropSrc
Points to an element whose properties are to be used as the properties of the referencing element. See Chapter 20, "Property and Class Definitions" on page 201.

Status
ignored by processes

Style
The Style attribute contains either a reference to a separate style specification

for an element or a set of element-specific presentation style definitions when the processing system does not support separate styles for elements. Assigning a value to the STYLE attribute modifies how an element is presented when it is output.

Usage

See “Creating a master index” on page 125.

Contexts

Children: empty.

Parents: MasterIndex.

MasterIndexPrefix (master index prefix)

Purpose

MasterIndexPrefix is used in conjunction with MasterIndexInfo. Use these tags when preparing to merge the indexes of several documents into a master index. These tags should be used in the content of each document containing an index you wish to have merged into a master index.

Examples

```
<masterindexinfo>
<masterindexprefix>GSUG</masterindexprefix>
</masterindexinfo>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Creating a master index” on page 125.

Contexts

Children: text (#pcdata).

Parents: MasterIndexInfo.

MD (marked deletion)

Purpose

The MD element identifies data that no longer applies. The REV attribute associates the MD element with a specific revision.

MD can contain any other phrase-like elements. The MD element cannot be used to mark data that contains paragraphs or division elements. In these cases, the revision and status attributes provided for those elements must be used to indicate the revision level and deletion status of the data.

Examples

```
The following data no longer applies:
<MD>This data no longer applies.</MD>,
as you can clearly see.
```

Attributes

Rev
The REV attribute references the Rev element which describes the last revision level of the information. See “Using Revisions” on page 109.

See “Common Element Attributes (large set)” on page 227.

Usage

See “Marking text for deletion” on page 111.

Contexts

Children: text (#pcdata), Address, APL, Bin, Char, Cit, Date, Dec, Formula, Hex, L, MV, Num, Oct, Ph, PK, PV, Q, RefKey, SynPh, Term, TM, XPh, XRef.

Parents: AnnotBody, Attention, Bridge, Caution, CompCmt, Danger, Defn, Desc, entry, Fn, L, LI, Lines, LQ, MkNote, ModDesc, ModItem, NoteBody, P, Ph, Q, SynNote, Warning.

MetaData (information architecture)

Purpose

The MetaData tag identifies or classifies your information. It is passed through to the XHTML output as metadata keywords. This helps search programs and other programs find, filter, or select information.

Examples

```
<d id="feederinst">
<dprolog><titleblk>
<title>Installing your Fruit-Bat Feeder</title>
</titleblk>
<metadata type="task" job="installing"
 audience="user" experiencelevel="general">
</dprolog>
...
```

Attributes

type
Indicates the information type. You can pick from: advisor, concept, definition, example, reference, task, or wizard. For information on information types, refer to UA Central: http://ua.raleigh.ibm.com/ua

job
Indicates the type of job being performed with the information. You can pick from: administering, configuring, customizing, evaluating, installing, planning, programming, troubleshooting, or using.

audience
Indicates the intended audience. You can pick from: administrator, executive, programmer, or user.

experiencelevel
Indicates how experienced the reader is. You can pick from: expert, general, or novice.

vrmoriginated
Indicates the version, release, and modification level that the information was originated in.

vrmlastchanged
Indicates the version, release, and modification level that the information was last updated.

classification
Indicates the classification of the information. You can pick classified or unclassified.

Usage

See "Creating an information architecture" on page 27.

Contexts

Children: empty.

Parents: Dprolog, SpecDprolog.

MkAction (marked note action definition)

Purpose

Use MkAction to define actions that can be associated with marked notes. You can define as many actions as you need. Any mark action can be used with any mark class. These actions are used within a MKNote element to describe the action that is associated with that note. The MarkList element uses the MKAction to determine which marked notes are to be presented.

See "Creating Collections of Marked Notes" on page 111.

Examples

```
<propdefs>
<mkdesc>
<mkclass name="msg">Msg</mkclass>
<mkclass name="abend">Abend</mkclass>
<mkaction name="new">New</mkaction>
<mkaction name="changed">Changed</mkaction>
<mkaction name="deleted">Deleted</mkaction>
<mkaction name="replaced">Replaced</mkaction>
</mkdesc>
</propdefs>
```

Attributes

NAME=*action_name*
Specifies the short name of the action, which is the value specified in the Action attribute of MkNote and the Actions attribute of MarkList.

Conloc
The CONLOC attribute specifies that the content of another element of the same type is to be used as the content of the referencing element. This enables reuse of information. When the CONLOC attribute is specified, you cannot

specify the element’s end tag. The result of using CONLOC is exactly the same as if the element being referred to had occurred at that point in the document. See “Reusing elements from an object library” on page 191.

Attributes on the element are now passed to the element with the CONLOC; this is a feature that began with IDWB release 3.4, patch IDWXF036.

ID=*identifier*
: The ID attribute identifies an element within an SGML document. IDs must be unique within a single document. Any element that has an ID can be cross-referenced or linked to. IDs can be up to 64 characters long. IDs must start with an alphabetic character and can contain letters, numbers, dashes (-), or periods (.).

Rev
: The REV attribute references the Rev element which describes the last revision level of the information. See “Using Revisions” on page 109.

Status
: ignored by processes

Style
: The Style attribute contains either a reference to a separate style specification for an element or a set of element-specific presentation style definitions when the processing system does not support separate styles for elements. Assigning a value to the STYLE attribute modifies how an element is presented when it is output.

HyTime
: ignored by processes

InfoMast
: A fixed attribute used to classify the element.

Reftype
: ignored by processes

Usage

See “Creating Collections of Marked Notes” on page 111.

Contexts

Children: text (#pcdata).

Parents: MkDesc.

MkClass (marked note class definition)

Purpose

The MkClass element defines a marked note class. These class codes are used in a MarkNote element to specify the class of the note.

Examples

```
<propdefs>
<mkdesc>
<mkclass name="msg">Msg</mkclass>
<mkclass name="abend">Abend</mkclass>
<mkaction name="new">New</mkaction>
<mkaction name="changed">Changed</mkaction>
```

```
<mkaction name="deleted">Deleted</mkaction>
<mkaction name="replaced">Replaced</mkaction>
</mkdesc>
</propdefs>
```

Attributes

NAME=*class_name*
Specifies the short name of the class, which is the value specified in the Class attribute of MkNote and the Classes attribute of MarkList.

Conloc
The CONLOC attribute specifies that the content of another element of the same type is to be used as the content of the referencing element. This enables reuse of information. When the CONLOC attribute is specified, you cannot specify the element's end tag. The result of using CONLOC is exactly the same as if the element being referred to had occurred at that point in the document. See "Reusing elements from an object library" on page 191.

Attributes on the element are now passed to the element with the CONLOC; this is a feature that began with IDWB release 3.4, patch IDWXF036.

ID=*identifier*
The ID attribute identifies an element within an SGML document. IDs must be unique within a single document. Any element that has an ID can be cross-referenced or linked to. IDs can be up to 64 characters long. IDs must start with an alphabetic character and can contain letters, numbers, dashes (-), or periods (.).

Rev
The REV attribute references the Rev element which describes the last revision level of the information. See "Using Revisions" on page 109.

Status
ignored by processes

Style
The Style attribute contains either a reference to a separate style specification for an element or a set of element-specific presentation style definitions when the processing system does not support separate styles for elements. Assigning a value to the STYLE attribute modifies how an element is presented when it is output.

HyTime
ignored by processes

InfoMast
A fixed attribute used to classify the element.

Reftype
ignored by processes

Usage

See "Creating Collections of Marked Notes" on page 111.

Contexts

Children: text (#pcdata).

Parents: MkDesc.

MkDesc (mark description)

Purpose

The MkDesc element describes the classes and actions that can be used with marked notes.

Use MkDesc to define the classes and actions that are valid for marked notes. Any mark action can be used with any mark class.

Examples

```
<propdefs>
<mkdesc>
<mkclass name="msg">Msg</mkclass>
<mkclass name="abend">Abend</mkclass>
<mkaction name="new">New</mkaction>
<mkaction name="changed">Changed</mkaction>
<mkaction name="deleted">Deleted</mkaction>
<mkaction name="replaced">Replaced</mkaction>
</mkdesc>
</propdefs>
```

Attributes

Conloc
The CONLOC attribute specifies that the content of another element of the same type is to be used as the content of the referencing element. This enables reuse of information. When the CONLOC attribute is specified, you cannot specify the element's end tag. The result of using CONLOC is exactly the same as if the element being referred to had occurred at that point in the document. See "Reusing elements from an object library" on page 191.

Attributes on the element are now passed to the element with the CONLOC; this is a feature that began with IDWB release 3.4, patch IDWXF036.

ID=*identifier*
The ID attribute identifies an element within an SGML document. IDs must be unique within a single document. Any element that has an ID can be cross-referenced or linked to. IDs can be up to 64 characters long. IDs must start with an alphabetic character and can contain letters, numbers, dashes (-), or periods (.).

Rev
The REV attribute references the Rev element which describes the last revision level of the information. See "Using Revisions" on page 109.

Status
ignored by processes

Style
The Style attribute contains either a reference to a separate style specification for an element or a set of element-specific presentation style definitions when the processing system does not support separate styles for elements. Assigning a value to the STYLE attribute modifies how an element is presented when it is output.

HyTime
ignored by processes

InfoMast
A fixed attribute used to classify the element.

Reftype
ignored by processes

Usage

See “Creating Collections of Marked Notes” on page 111.

Contexts

Children: MkAction, MkClass.

Parents: PropDefs, PropGroup.

MkNote (marked note)

Purpose

The MkNote element contains a marked note, which is a specialized annotation element that can be associated with problem and change tracking information such as change requests or problem numbers. Use marked notes to collect information about the content of your document such as notes about changes, notes to yourself as an author, or references to specific locations as navigation aides to readers. Marked notes are presented in marked note lists as defined by the MarkList element.

For example, you can automatically create a summary of changes by using marked notes to record changes within your document and using a MarkList element to generate a list of those notes.

Examples

```
<ibmiddoc docstyle="ibmxagd">
<prolog><ibmbibentry><doctitle><titleblk>
<title>My Marked Changes Document for Messages</title>
</titleblk></doctitle></ibmbibentry>
<propdefs>
<mkdesc>
<!--Define two classes for marked lists - notes and abends-->
<mkclass name="msg">Msg</mkclass>
<mkclass name="abend">Abend</mkclass>
<!--Define the actions for the changed info-->
<mkaction name="new">New</mkaction>
<mkaction name="change">Changed</mkaction>
<mkaction name="del">Deleted</mkaction>
<mkaction name="rep">Replaced</mkaction>
</mkdesc>
</propdefs>
<revdefs>
<rev id="revv4r5" ident="use">
<date></date>
<desc></desc>
</rev>
<mark id="mkv4r5" ident="use">
<desc>v4r5 marked message changes</desc>
</mark>
</revdefs>
</prolog>
<body>
<d>
```

```
<dprolog><titleblk>
<title>List of changed items</title>
</titleblk></dprolog>
<dbody>
<marklist mkids="mkv4r5" classes="msg abend" actions="change del rep"
display="item action page desc" classhd="Msg" actionhd="Reason"
itemhd="Msg" lochd="Page" deschd="Message text"></dbody>
</d>
<msglist>
<msg rev="revv4r5">
<msgnum>IDW0012</msgnum>
<msgtext>Hi there!</msgtext>
<msgitem class="xpl">
<p>This is a friendly message.<mknote class="msg"
action="change" mkids="mkv4r5" item="IDW0012">Hi there!
</mknote></p>
</msgitem>
</msg>
<msg rev="revv4r5">
<msgnum>IDW0013</msgnum>
<msgtext>Farewell!</msgtext>
<msgitem class="xpl">
<p>This unlucky message was removed.<mknote class="msg"
action="del" mkids="mkv4r5" item="IDW0013">Farewell!
</mknote></p>
</msgitem>
</msg>
</msglist></body>
</ibmiddoc>
```

Attributes

CLASS=*class_name*
Defines one or more mark classes to which this marked note belongs.

ACTION=*action_name*
Defines one or more actions associated with this marked note.

MKIDS=*mark_id*
Contains the IDs of one or more Mark elements, which define collections of marked notes.

ITEM=*item_value*
Defines an identifying label for the note, such as a message number, function name, or error report. The Item attribute enables you to distinguish among marks of the same class and action. You can also use the Item attribute to closely associate a note with things in your document that have unique identifiers such as message numbers.

DISPLAY=*items-to-display*
Specifies the items to display and the order to display them in. Values are: item, action, loc or page, and desc.

CLASSHD=*class-heading*
Specifies the heading for the Class column.

ACTIONHD=*action-heading*
Specifies the heading for the Action column.

ITEMHD=*item-heading*
Specifies the heading for the Item column.

LOCHD=*location-heading*
Specifies the heading for the Location column.

DESCHD=*description-heading*
Specifies the heading for the Description column.

See "Common Element Attributes (large set)" on page 227.

Usage

See "Creating Collections of Marked Notes" on page 111.

Contexts

Children: text (#pcdata), Address, Annot, APL, Bin, Bridge, CGraphic, Char, Cit, Date, Dec, DL, Fig, Formula, GL, Hex, L, Lines, LitData, MD, MMObj, MV, Note, NoteList, Num, Oct, OL, P, ParmL, PBlk, Ph, PK, PV, Q, RefKey, Screen, SynPh, Syntax, Term, TM, UL, Xmp, XPh, XRef.

Parents: AnnotBody, Attention, Bridge, Caution, Danger, DBody, Defn, DIntro, DSum, entry, Fig, FigSeg, LEDI, LI, LQ, ModDesc, ModItem, MsgItem, NoteBody, P, PBlk, ProcIntro, Warning.

MMObj (multi-media object; artwork)

Purpose

The MMObj element refers to a non-text object such as an image, vector graphic, or video clip. The types of objects supported depends on your processing and presentation systems. For print, non-text objects are usually image or vector graphics such as EPS graphics. For online information, they would be BMPs, GIFs, or JPEGs. MMObj also contains a text description of the object, intended to provide an alternative to the object itself.

Depending on the style definition, the object can be integrated with the text. It is normally presented inline with the text. The online presentation of non-text objects depends on the presentation system being used. When you need to refer to a single non-text object, you can use the OBJ attribute of the MMObj element to specify the entity that represents the object.

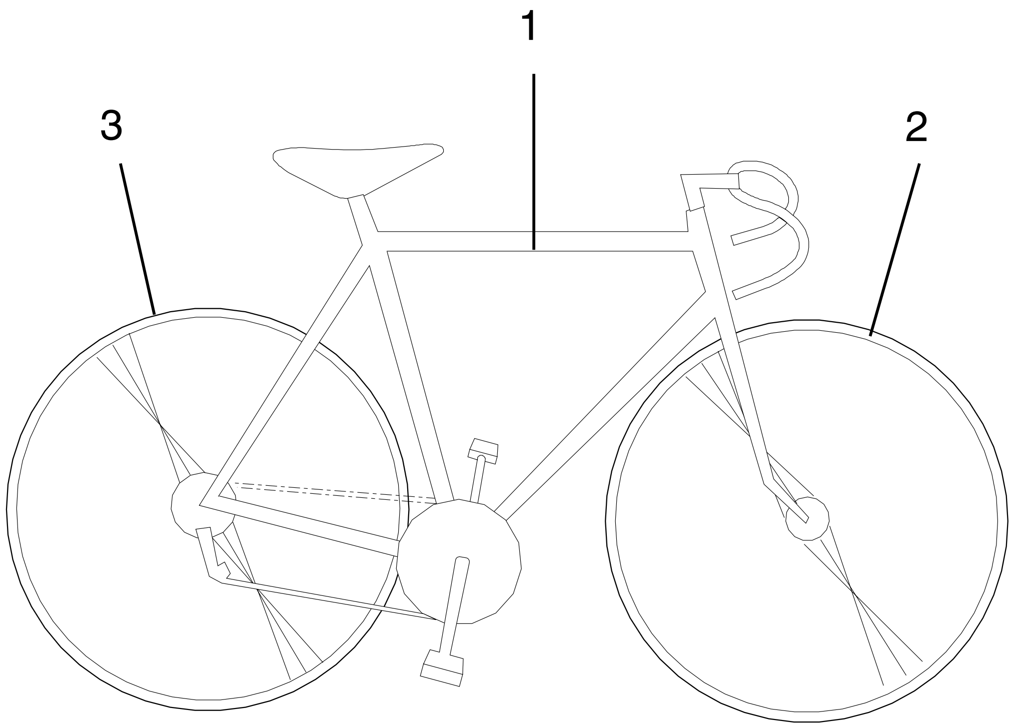
Migration Note

MMObj replaces the ARTWORK and ARTALT tags. Use MMObj as you would ARTALT. The presentation attributes on the ARTWORK tag are now notation attributes specified on the entity declaration used to declare the artwork file.

Examples

Here's the declaration and the markup for an illustration of a bike:

```
<!ENTITY bike system "bike.gif" ndata graphics>
  ...
<MMOBJ>
<OBJREF OBJ="bike">
<TEXTALT>This is a two-wheeled bicycle.</TEXTALT>
</MMOBJ>
```



Attributes

LABEL=*position*
: Defines a text label to use for a graphic object when the object cannot be presented, or to use as a link button.

Placement=*position*
: Controls where the grapic appears on the page. Valid values include: standalone, inline, and margin.

  Standalone
  : Places artwork on a sparate line. This is the default value.

  Inline
  : Places artwork in the current text stream without a preceding or following line break. This replaces the previously used style override `bkm:(runin)`

  Margin
  : | Places artwork in the offset margin. In some Xyvision-formatted styles, the
  | placement does not work as expected.

Halign=*alignment*
: Controls the horizontal alignment of the graphic witin the textline when standalone is specified. Valid values include: left, center, right, and current.

See "Common Element Attributes (large set)" on page 227.

Usage

See "Including artwork in documents" on page 55.

Contexts

Children: MMObjLink, ObjRef, TextAlt.

MMObj (multi-media object; artwork)

Parents: AnnotBody, Attention, BackCover, Bridge, Caution, Cond, Copyr, CoverDef, Danger, DBody, Defn, Desc, DIntro, DSum, EdNotices, entry, Fig, FigSeg, Fn, FrontCover, LEDesc, LEDI, LI, LQ, MkNote, ModDesc, ModItem, MsgItem, NItem, NoteBody, Notices, P, PartAsm, PartAsmSeg, PBlk, ProcEntry, ProcExit, ProcIntro, Safety, Screen, Sem, SynNote, Term, Warning.

MMObjLink (multi-media object link)

Purpose

The MMObjLink element allows you to create a graphic hot spot under an image.

Examples

```
<mmobj><objref obj="part1">
<mmobjlink linkend="newdiv">
<areadef coords="1 100">
</areadef></mmobjlink>
<textalt></textalt>
</mmobj>
```

Attributes

LINKEND=*text_target*
Specifies the ID of the textual target of the link.

AREADEFS=*ref-ID*
Specifies the ID of the AreaDef element containing the graphic area specification of the graphic hot spot. If AREADEFS is specified, MMObjLink must be an empty element. To specify the area definition, see “AreaDef (defines graphic hot spot area)” on page 236.

If you do not specify an area definition using an ID, use an inline AreaDefs element to define the hotspot.

See “Common Element Attributes (large set)” on page 227.

Usage

See “Creating graphic links” on page 56.

Contexts

Children: AreaDef.

Parents: MMObj.

Mod (information module)

Purpose

The Mod element contains a module of information within a collection of modules. The modules usually have the same structure.

Use modular information to create reference information. You can define as many different modular information classes as you want for information that is similarly structured.

Examples

```
<modinfo class="cust" style="table">
<mod class="CUST">
<modname class="NAME">Fred Smith</modname>
<moditem class="CUSTID"><p><num base="10">1000</num></p></moditem>
<moditem class="INC"><num base="10">40000</num></moditem>
<moditem class="LPD"><p><date>12/25/93</date></p></moditem>
<moditem class="NOTES"><p>Big spender</p></moditem>
</mod>
<mod class="CUST">
<modname class="NAME">Suzanne Stanley</modname>
<moditem class="CUSTID"><p><num base="10">1001</num></p></moditem>
<moditem class="INC"><p><num base="10">50000</num></p></moditem>
<moditem class="LPD"><p><date>11/22/92</date></p></moditem>
<moditem class="NOTES"><p>Likes game software</p></moditem>
</mod>
<mod class="CUST">
<modname class="NAME">Jeff George</modname>
<moditem class="CUSTID"><p><num base="10">1002</num></p></moditem>
<moditem class="INC"><p><num base="10">60000</num></p></moditem>
<moditem class="LPD"><p><date>12/02/93</date></p></moditem>
<moditem class="NOTES"><p>Likes DVD movies</p></moditem>
</mod>
<mod class="CUST">
<modname class="NAME">Mike Gidento</modname>
<moditem class="CUSTID"><p><num base="10">1003</num></p></moditem>
<moditem class="INC"><p><num base="10">35000</num></p></moditem>
<moditem class="LPD"><p><date>12/12/92</date></p></moditem>
<moditem class="NOTES"><p>Likes 8-track tapes</p></moditem>
</mod>
</modinfo>
```

Attributes

CLASS=*classname*
Defines the class of this information module. Classes are defined with a ModInfoDef element within the document or division prolog.

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 17, “Defining Modular Information” on page 175.

Contexts

Children: ModDesc, ModItem, ModName, RetKey.

Parents: ModInfo.

ModDesc (modular content description)

Purpose

The ModDesc element contains a description of the content of the Mod element of which it is a part.

Examples

```
<modinfo class="cust" style="table">
<mod class="CUST">
<modname class="NAME">Fred Smith</modname>
```

ModDesc (modular content description)

```
<moddesc>Fred is a cool guy</moddesc>
<moditem class="CUSTID"><p><num base="10">1000</num></p></moditem>
<moditem class="INC"><num base="10">40000</num></moditem>
<moditem class="LPD"><p><date>12/25/93</date></p></moditem>
<moditem class="NOTES"><p>Big spender</p></moditem>
</mod>
<mod class="CUST">
<modname class="NAME">Suzanne Stanley</modname>
<moditem class="CUSTID"><p><num base="10">1001</num></p></moditem>
<moditem class="INC"><p><num base="10">50000</num></p></moditem>
<moditem class="LPD"><p><date>11/22/92</date></p></moditem>
<moditem class="NOTES"><p>Likes game software</p></moditem>
</mod>
<mod class="CUST">
<modname class="NAME">Jeff George</modname>
<moditem class="CUSTID"><p><num base="10">1002</num></p></moditem>
<moditem class="INC"><p><num base="10">60000</num></p></moditem>
<moditem class="LPD"><p><date>12/02/93</date></p></moditem>
<moditem class="NOTES"><p>Likes DVD movies</p></moditem>
</mod>
<mod class="CUST">
<modname class="NAME">Mike Gidento</modname>
<moditem class="CUSTID"><p><num base="10">1003</num></p></moditem>
<moditem class="INC"><p><num base="10">35000</num></p></moditem>
<moditem class="LPD"><p><date>12/12/92</date></p></moditem>
<moditem class="NOTES"><p>Likes 8-track tapes</p></moditem>
</mod>
</modinfo>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 17, “Defining Modular Information” on page 175.

Contexts

Children: text (#pcdata), Address, Annot, APL, Attention, Bin, Bridge, Caution, CGraphic, Char, Cit, Danger, Date, Dec, DL, Fig, Formula, GL, Hex, L, Lines, LitData, LQ, MD, MkNote, MMObj, MV, Note, NoteList, Num, Oct, OL, P, ParmL, PBlk, Ph, PK, PV, Q, RefKey, Screen, SynPh, Syntax, Term, TM, UL, Xmp, XPh, XRef.

Parents: Mod.

ModInfo (modular information)

Purpose

The ModInfo element contains a set of information modules for a single class of modular information.

Use a ModInfo element to contain a set of information modules that describe a type of information that is specific to your document. For example, in a document describing a complex program, you can define a class of information module for describing data structures.

Note: If your information describes the elements of a computer language, use the LERS element for your information.

Information modules can also be organized within object libraries and used by reference from within a modular information section using the CONLOC attribute.

The default presentation style for modular information is as divisions, but other presentation styles are defined. For example, you can present each module as a row in a table for quick reference.

MODINFO with any style except STYLE=TABLE becomes a series of nested divisions. When STYLE=TABLE is used, it is mapped to a table. The title of MODINFO becomes CAP; DESC becomes DESC; MODINFOTITLE becomes ENTRY; MOD becomes ROW; MODNAME becomes ENTRY, and the TITLE in MODITEM is suppressed.

To use information modules, you must define an information module class using the ModInfoDef element. To better facilitate reuse, you should consider containing the ModInfoDef element within your modular information unit. You can specify the modular information class on the ModInfo element, which applies to all contained Mod elements, or you can specify a class on each Mod element.

For example, suppose you define several related classes of information module that you want to group into a single ModInfo group for presentation. In that case, you need to specify the class for each module rather than a global class for the ModInfo element.

Examples

```
<modinfo class="cust" style="table">
<mod class="CUST">
<modname class="NAME">Fred Smith</modname>
<moditem class="CUSTID"><p><num base="10">1000</num></p></moditem>
<moditem class="INC"><num base="10">40000</num></moditem>
<moditem class="LPD"><p><date>12/25/93</date></p></moditem>
<moditem class="NOTES"><p>Big spender</p></moditem>
</mod>
<mod class="CUST">
<modname class="NAME">Suzanne Stanley</modname>
<moditem class="CUSTID"><p><num base="10">1001</num></p></moditem>
<moditem class="INC"><p><num base="10">50000</num></p></moditem>
<moditem class="LPD"><p><date>11/22/92</date></p></moditem>
<moditem class="NOTES"><p>Likes game software</p></moditem>
</mod>
<mod class="CUST">
<modname class="NAME">Jeff George</modname>
<moditem class="CUSTID"><p><num base="10">1002</num></p></moditem>
<moditem class="INC"><p><num base="10">60000</num></p></moditem>
<moditem class="LPD"><p><date>12/02/93</date></p></moditem>
<moditem class="NOTES"><p>Likes DVD movies</p></moditem>
</mod>
<mod class="CUST">
<modname class="NAME">Mike Gidento</modname>
<moditem class="CUSTID"><p><num base="10">1003</num></p></moditem>
<moditem class="INC"><p><num base="10">35000</num></p></moditem>
<moditem class="LPD"><p><date>12/12/92</date></p></moditem>
<moditem class="NOTES"><p>Likes 8-track tapes</p></moditem>
</mod>
</modinfo>
```

Attributes

style=table
Causes the modular information to be presented in a table format. The default is as nested headings.

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 17, “Defining Modular Information” on page 175.

Contexts

Children: Desc, Mod, ModInfoDef, RetKey, Title.

Parents: AnnotBody, Appendix, Attention, Body, Bridge, Caution, Danger, DBody, Defn, DIntro, DSum, Fn, LEDI, LI, LQ, MsgItem, NoteBody, P, PBlk, ProcIntro, SynNote, Warning.

ModInfoDef (modular information property definition)

Purpose

The ModInfoDef element defines a set of modular information properties.

Use ModInfoDef to define a class of modular information specific to your information. You can apply specific presentation styles to a given class of modular information using the STYLE attribute.

All classes referenced by a ModInfo element must be defined. These classes are usually defined in a central location that is accessed by many documents. This centralization allows control over the class definitions for modular information used at a publishing site.

The scope of a definition is determined by the location of the definition.

Examples

```
<propdefs>
<modinfodef classname="CUST"><desc>Customer information
</desc>
<moditemdef classname="NAME"><title>Name</title><desc>
This is the first and last name of the customer</desc>
</moditemdef>
<moditemdef classname="CUSTID"><title>ID</title><desc>
This is their customer id number</desc></moditemdef>
<moditemdef classname="INC"><title>Income</title>
<desc>This is their annual estimated income</desc>
</moditemdef>
<moditemdef classname="LPD"><title>Last purchase date
</title><desc>This is their last purchase date</desc>
</moditemdef>
<moditemdef classname="NOTES"><title>Notes</title>
<desc>Personal notes</desc></moditemdef>
</modinfodef>
</propdefs>
```

Attributes

CLASSNAME=*classname*
: Specifies the class of modular information that is being defined.

Usage

See Chapter 17, “Defining Modular Information” on page 175.

Contexts

Children: Desc, ModItemDef.

Parents: ModInfo, PropDefs, PropGroup.

ModItemDef (item class definitions)

Purpose

The ModItemDef element defines the classes of module description items valid for a class of modular information.

Examples

```
<propdefs>
<modinfodef classname="CUST"><desc>Customer information
</desc>
<moditemdef classname="NAME"><title>Name</title><desc>
This is the first and last name of the customer</desc>
</moditemdef>
<moditemdef classname="CUSTID"><title>ID</title><desc>
This is their customer id number</desc></moditemdef>
<moditemdef classname="INC"><title>Income</title>
<desc>This is their annual estimated income</desc>
</moditemdef>
<moditemdef classname="LPD"><title>Last purchase date
</title><desc>This is their last purchase date</desc>
</moditemdef>
<moditemdef classname="NOTES"><title>Notes</title>
<desc>Personal notes</desc></moditemdef>
</modinfodef>
</propdefs>
```

Attributes

CLASSNAME=*classname*
: Defines the class of module description item that is valid for a class of modular information.

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 17, “Defining Modular Information” on page 175.

Contexts

Children: Desc, Title.

Parents: ModInfoDef, PropDefs, PropGroup.

ModItem (module description item)

Purpose

The ModItem element contains one item in a module of information.

The ModItem elements within a Mod element contain the different kinds of information that are applicable for a given class of information module. The classes for the ModItem elements are defined in the ModItemDef element.

The style used to present ModItems is determined by the style specification of the Mod element which contains it.

Examples

```
<modinfo class="cust" style="table">
<mod class="CUST">
<modname class="NAME">Fred Smith</modname>
<moditem class="CUSTID"><p><num base="10">1000</num></p></moditem>
<moditem class="INC"><num base="10">40000</num></moditem>
<moditem class="LPD"><p><date>12/25/93</date></p></moditem>
<moditem class="NOTES"><p>Big spender</p></moditem>
</mod>
<mod class="CUST">
<modname class="NAME">Suzanne Stanley</modname>
<moditem class="CUSTID"><p><num base="10">1001</num></p></moditem>
<moditem class="INC"><p><num base="10">50000</num></p></moditem>
<moditem class="LPD"><p><date>11/22/92</date></p></moditem>
<moditem class="NOTES"><p>Likes game software</p></moditem>
</mod>
<mod class="CUST">
<modname class="NAME">Jeff George</modname>
<moditem class="CUSTID"><p><num base="10">1002</num></p></moditem>
<moditem class="INC"><p><num base="10">60000</num></p></moditem>
<moditem class="LPD"><p><date>12/02/93</date></p></moditem>
<moditem class="NOTES"><p>Likes DVD movies</p></moditem>
</mod>
<mod class="CUST">
<modname class="NAME">Mike Gidento</modname>
<moditem class="CUSTID"><p><num base="10">1003</num></p></moditem>
<moditem class="INC"><p><num base="10">35000</num></p></moditem>
<moditem class="LPD"><p><date>12/12/92</date></p></moditem>
<moditem class="NOTES"><p>Likes 8-track tapes</p></moditem>
</mod>
</modinfo>
```

Attributes

CLASS=*classname*
: Indicates which class of information the ModItem contains. The class values for a modular information item are defined on the ModIntemDef element.

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 17, “Defining Modular Information” on page 175.

Contexts

Children: text (#pcdata), Address, Annot, APL, Attention, Bin, Bridge, Caution, CGraphic, Char, Cit, Danger, Date, Dec, DL, Fig, Formula, GL, Hex, L, LERS, Lines, LitData, LQ, MD, MkNote, MMObj, MsgList, MV, Note, NoteList, Num, Oct, OL, P, ParmL, PartAsm, PBlk, Ph, PK, Proc, PV, Q, RefKey, Screen, SynPh, Syntax, Term, TM, UL, Xmp, XPh, XRef.

Parents: Mod.

ModLvl (modification level)

Purpose

The ModLvl element contains a program's modification level.

Examples

```
<IBMPRODINFO>
<PRODNAME>Test Prod</PRODNAME>
<VERSION>2</VERSION>
<RELEASE>3</RELEASE>
<MODLVL>1</MODLVL>
<IBMPROGNUM>223-3330</IBMPROGNUM>
</IBMPRODINFO>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See "Other prolog elements" on page 90.

Contexts

Children: text (#pcdata), Ph.

Parents: IBMProdInfo.

ModName (modular information element name)

Purpose

The ModName element contains the name of the information contained by the Mod element of which it is a part.

Examples

```
<modinfo class="cust" style="table">
<mod class="CUST">
<modname class="NAME">Fred Smith</modname>
<moditem class="CUSTID"><p><num base="10">1000</num></p></moditem>
<moditem class="INC"><num base="10">40000</num></moditem>
<moditem class="LPD"><p><date>12/25/93</date></p></moditem>
<moditem class="NOTES"><p>Big spender</p></moditem>
</mod>
<mod class="CUST">
<modname class="NAME">Suzanne Stanley</modname>
<moditem class="CUSTID"><p><num base="10">1001</num></p></moditem>
<moditem class="INC"><p><num base="10">50000</num></p></moditem>
<moditem class="LPD"><p><date>11/22/92</date></p></moditem>
<moditem class="NOTES"><p>Likes game software</p></moditem>
</mod>
<mod class="CUST">
<modname class="NAME">Jeff George</modname>
```

```
<moditem class="CUSTID"><p><num base="10">1002</num></p></moditem>
<moditem class="INC"><p><num base="10">60000</num></p></moditem>
<moditem class="LPD"><p><date>12/02/93</date></p></moditem>
<moditem class="NOTES"><p>Likes DVD movies</p></moditem>
</mod>
<mod class="CUST">
<modname class="NAME">Mike Gidento</modname>
<moditem class="CUSTID"><p><num base="10">1003</num></p></moditem>
<moditem class="INC"><p><num base="10">35000</num></p></moditem>
<moditem class="LPD"><p><date>12/12/92</date></p></moditem>
<moditem class="NOTES"><p>Likes 8-track tapes</p></moditem>
</mod>
</modinfo>
```

Attributes

LINESPACE=SPACE | COMPACT
Specifies whether the items in the list should be compacted or have space between the items. Nested lists automatically inherit the setting of the outer list, but can override that default with their own setting.

| **Layout=Default-Layout | TwoCol | Page**
| This specifies how the message list should be formatted.

| **Default-Layout**
| Uses the layout based on the style. Most of the styles will have the default
| as two columns, but some of the smaller styles have a default of
| page-wide.

| **TwoCol**
| Composes the message list in two columns.

| **Page**
| Composes the message list page-wide.

| **RETKEY=None | NoDup**
Use the RetKey attribute to enable or disable automatic running headings for this tag.

NONE
indicates that nothing is to be used

NODUP
indicates that the first and last non-blank items on the page are to be used. The two items are joined together, separated by a bullet or other character, and the combined string is used as the retrieval subject for the page. Use of NODUP provides the most dictionary-like retrieval of items. If the first and last items on the page are the same, only one of the items appears.

If you code any explicit RetKey elements, they are honored and will appear. If you nest elements that can generate a running heading (for example, a MsgList inside Lers), only the outer active generated heading is used. That is, if you specified automated RetKey generation for LERS and MSGLIST, a Msgno inside Lers will not be used in the RetKey area. But if you had an explicit RetKey inside the Msg, then the RetKey is honored as an explicit override.

Props=*properties*
The Props attribute is used to specify the condition under which the information contained within the element appears. See “Property-Based Retrieval” on page 195.

| **ScalePct=***scale-percent*
| You can use the ScalePct attribute to scale the text up or down in the element.

| The scale-percent is a positive, whole number. 100 is the normal size; 50 is 50%
| smaller; 200 is 200% larger. For example, this scales the text to 80% of the body
| text:

```
scalepct="80"
```

See "Common Element Attributes (large set)" on page 227.

Usage

For more information about the ModName element, see "Examples of Using Modular Information" on page 176.

Contexts

Children: text (#pcdata), L, Ph, Term, TM.

Parents: Mod.

Msg (message or code description)

Purpose

The Msg element contains a message or code and its description.

Examples

```
<msglist>
<msg>
<msgnum>DJI7832E</msgnum>
<msgtext>This message is issued when no data set of
the name <mv>file-name</mv> is found.</msgtext>
<msgitem class="xpl">
<p>The processor could not locate the data set named <mv>
file-name</mv>.</p>
</msgitem>
<msgitem class="severity">
<p>8</p>
</msgitem>
<msgitem class="probd">
<p>You would appear to have a problem.</p>
</msgitem>
<msgitem class="uresp">
<p>Search high and low for the data set.</p>
</msgitem>
</msg>
<msg>
<msgtext>This message has no number</msgtext>
<msgitem class="xpl">
<p>This message has no message number; only text.
These are really insidious because it makes finding
the message very hard.</p>
</msgitem>
</msg>
</msglist>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See "Message and code lists" on page 38.

Msg (message or code description)

Contexts

Children: Code, MsgItem, MsgNum, MsgText, Title.

Parents: MsgList.

MsgItem (message description item)

Purpose

The MsgItem element contains part of the description of a message or code.

Examples

```
<msglist>
<msg>
<msgnum>DJI7832E</msgnum>
<msgtext>This message is issued when no data set of
the name <mv>file-name</mv> is found.</msgtext>
<msgitem class="xpl">
<p>The processor could not locate the data set named <mv>
file-name</mv>.</p>
</msgitem>
<msgitem class="severity">
<p>8</p>
</msgitem>
<msgitem class="probd">
<p>You would appear to have a problem.</p>
</msgitem>
<msgitem class="uresp">
<p>Search high and low for the data set.</p>
</msgitem>
</msg>
<msg>
<msgtext>This message has no number</msgtext>
<msgitem class="xpl">
<p>This message has no message number; only text.
These are really insidious because it makes finding
the message very hard.</p>
</msgitem>
</msg>
</msglist>
```

Attributes

CLASS=*author-defined_class* **| DEST | XPL | EXPLANATION | MODULE | NUMBYTES | ORESP | PRESP | PROBD | SEVERITY | SPRESP | SYSACT | URESP**

Indicates the class of the MsgItem as follows:

author-defined_class
: Specifies an author-defined class, which must be defined using a MsgItemDef element.

DEST
: Specifies the message destination.

XPL or EXPLANATION
: Specifies the message explanation.

MODULE
: Specifies the issuing module.

NUMBYTES
: Specifies the number of error bytes.

ORESP
: Specifies the operator response.

PRESP
: Specifies the programmer response.

PROBD
: Specifies problem-determination information.

SEVERITY
: Specifies the message severity.

SPRESP
: Specifies the system-programmer response.

SYSACT
: Specifies the system action.

URESP
: Specifies the user response.

Usage

See "Message and code lists" on page 38.

Contexts

Children: Annot, AsmList, Attention, BibList, Bridge, Caution, CGraphic, D, Danger, DBlk, DL, Fig, FnList, GL, L, LERS, Lines, LitData, LQ, MarkList, MkNote, MMObj, ModInfo, MsgList, Note, NoteList, OL, P, ParmL, PartAsm, PBlk, Proc, Screen, Syntax, Table, UL, Xmp.

Parents: Msg.

MsgItemDef (definition of message description items)

Purpose

The MsgItemDef element defines the classes of message description items that are valid for a message list.

Examples

```
<msglist>
<msgitemdef classname="xpl"><title>Why</title></msgitemdef>
<msgitemdef classname="uresp"><title>What to do</title>
</msgitemdef>
<msg>
<msgnum>A12</msgnum>
<msgtext>Closet full: Insufficient storage to proceed.
</msgtext>
<msgitem class="xpl">
<p>There are too many clothes in the closet.</p>
</msgitem>
<msgitem class="uresp">
<p>Remove some clothes from the closet and restart.
</p>
</msgitem>
</msg>
</msglist>
```

Attributes

CLASSNAME=*author-defined_class* **| DEST | EXPLANATION | MODULE | NUMBYTES | ORESP | PRESP | PROBD | SEVERITY | SPRESP | SYSACT | URESP**
Specifies the MsgItem class to which the definition applies as follows:

author-defined_class
Specifies an author-defined class, which must be defined using a MsgItemDef element.

DEST
Specifies the message destination.

EXPLANATION
Specifies the message explanation.

MODULE
Specifies the issuing module.

NUMBYTES
Specifies the number of error bytes.

ORESP
Specifies the operator response.

PRESP
Specifies the programmer response.

PROBD
Specifies problem-determination information.

SEVERITY
Specifies the message severity.

SPRESP
Specifies the system-programmer response.

SYSACT
Specifies the system action.

URESP
Specifies the user response.

Usage

See "Message and code lists" on page 38.

Contexts

Children: Desc, Title.

Parents: MsgList, PropDefs, PropGroup.

MsgLDef (Message list definition)

Purpose

The MsgLDef element sets attribute defaults for message lists. MsgLDef goes within the document prolog to set definitions for the entire document; or within a division prolog to set definitions for just that division. The MsgLDef tag goes inside a PropDefs tag.

Examples

```
<propdefs>
<msgldef defname="msginline" layout="default-layout" retkey="none">
</propdefs>
```

Attributes

DEFNAME=*definition-name*
The DefName attribute identifies a definition element. DefName values must be unique within a single document. A element that has a DEF attribute can point a corresponding tag with a DefName attribute. DefName values can be up to 64 characters long. DefName values must start with an alphabetic character and can contain letters, numbers, dashes (-), or periods (.).

Definition tags can be specified without a DefName attribute in the document's prolog. This allows you to change the initial settings of attributes for the entire document when the DEF tag is in the document's prolog. In a division's prolog, it changes the initial settings for that division.

LINESPACE=SPACE | COMPACT
Specifies whether the items in the list should be compacted or have space between the items. Nested lists automatically inherit the setting of the outer list, but can override that default with their own setting.

Layout=Default-Layout | TwoCol | Page
This specifies how the message list should be formatted.

Default-Layout
Uses the layout based on the style. Most of the styles will have the default as two columns, but some of the smaller styles have a default of page-wide.

TwoCol
Composes the message list in two columns.

Page
Composes the message list page-wide.

RETKEY=None | NoDup
Use the RetKey attribute to enable or disable automatic running headings for this tag.

NONE
indicates that nothing is to be used

NODUP
indicates that the first and last non-blank items on the page are to be used. The two items are joined together, separated by a bullet or other character, and the combined string is used as the retrieval subject for the page. Use of NODUP provides the most dictionary-like retrieval of items. If the first and last items on the page are the same, only one of the items appears.

If you code any explicit RetKey elements, they are honored and will appear. If you nest elements that can generate a running heading (for example, a MsgList inside Lers), only the outer active generated heading is used. That is, if you specified automated RetKey generation for LERS and MSGLIST, a Msgno inside Lers will not be used in the RetKey area. But if you had an explicit RetKey inside the Msg, then the RetKey is honored as an explicit override.

Props=*properties*
The Props attribute is used to specify the condition under which the information contained within the element appears. See “Property-Based Retrieval” on page 195.

ScalePct=*scale-percent*
You can use the ScalePct attribute to scale the text up or down in the element. The scale-percent is a positive, whole number. 100 is the normal size; 50 is 50% smaller; 200 is 200% larger. For example, this scales the text to 80% of the body text:

```
scalepct="80"
```

Usage

See Chapter 9, “Using definition tags” on page 105.

Contexts

Children: empty.

Parents: PropDefs, PropGroup.

MsgList (list of message or code descriptions)

Purpose

Use MsgList to contain descriptions of messages and codes. A MsgList contains one or more Msg elements, which contain the message or code and any explanatory text associated with it.

Examples

```
<msglist>
<msg>
<msgnum>DJI7832E</msgnum>
<msgtext>This message is issued when no data set of
the name <mv>file-name</mv> is found.</msgtext>
<msgitem class="xpl">
<p>The processor could not locate the data set named <mv>
file-name</mv>.</p>
</msgitem>
<msgitem class="severity">
<p>8</p>
</msgitem>
<msgitem class="probd">
<p>You would appear to have a problem.</p>
</msgitem>
<msgitem class="uresp">
<p>Search high and low for the data set.</p>
</msgitem>
</msg>
<msg>
<msgtext>This message has no number</msgtext>
<msgitem class="xpl">
<p>This message has no message number; only text.
These are really insidious because it makes finding
the message very hard.</p>
</msgitem>
</msg>
</msglist>
```

Attributes

DEF=*definition-name*
: Specifies the definition to be used. This points to a DefName attribute on a corresponding DEF tag. The settings defined on that DEF tag are used as defaults for this tag. See Chapter 9, “Using definition tags” on page 105 for more information.

Layout=Default-Layout | TwoCol | Page
: This specifies how the message list should be formatted.

Default-Layout
: Uses the layout based on the style. Most of the styles will have the default as two columns, but some of the smaller styles have a default of page-wide.

TwoCol
: Composes the message list in two columns.

Page
: Composes the message list page-wide.

RETKEY=None | NoDup
: Use the RetKey attribute to enable or disable automatic running headings for this tag.

NONE
: indicates that nothing is to be used

NODUP
: indicates that the first and last non-blank items on the page are to be used. The two items are joined together, separated by a bullet or other character, and the combined string is used as the retrieval subject for the page. Use of NODUP provides the most dictionary-like retrieval of items. If the first and last items on the page are the same, only one of the items appears.

If you code any explicit RetKey elements, they are honored and will appear. If you nest elements that can generate a running heading (for example, a MsgList inside Lers), only the outer active generated heading is used. That is, if you specified automated RetKey generation for LERS and MSGLIST, a Msgno inside Lers will not be used in the RetKey area. But if you had an explicit RetKey inside the Msg, then the RetKey is honored as an explicit override.

ScalePct=*scale-percent*
: You can use the ScalePct attribute to scale the text up or down in the element. The scale-percent is a positive, whole number. 100 is the normal size; 50 is 50% smaller; 200 is 200% larger. For example, this scales the text to 80% of the body text:

```
scalepct="80"
```

See “Common Element Attributes (large set)” on page 227.

Usage

See “Message and code lists” on page 38.

Contexts

Children: Msg, MsgItemDef, RetKey.

Parents: Appendix, Body, DBody, LEDI, ModItem, MsgItem.

MsgNum (message identifier)

Purpose

The MsgNum element contains the number of a message.

Examples

```
<msglist>
<msg>
<msgnum>DJI7832E</msgnum>
<msgtext>This message is issued when no data set of
the name <mv>file-name</mv> is found.</msgtext>
<msgitem class="xpl">
<p>The processor could not locate the data set named <mv>
file-name</mv>.</p>
</msgitem>
<msgitem class="severity">
<p>8</p>
</msgitem>
<msgitem class="probd">
<p>You would appear to have a problem.</p>
</msgitem>
<msgitem class="uresp">
<p>Search high and low for the data set.</p>
</msgitem>
</msg>
<msg>
<msgtext>This message has no number</msgtext>
<msgitem class="xpl">
<p>This message has no message number; only text.
These are really insidious because it makes finding
the message very hard.</p>
</msgitem>
</msg>
</msglist>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See "Message and code lists" on page 38.

Contexts

Children: text (#pcdata), Ph.

Parents: Msg.

MsgText (message text)

Purpose

The MsgText element contains the text of a message.

Examples

```
<msglist>
<msg>
<msgnum>DJI7832E</msgnum>
<msgtext>This message is issued when no data set of
```

```
the name <mv>file-name</mv> is found.</msgtext>
<msgitem class="xpl">
<p>The processor could not locate the data set named <mv>
file-name</mv>.</p>
</msgitem>
<msgitem class="severity">
<p>8</p>
</msgitem>
<msgitem class="probd">
<p>You would appear to have a problem.</p>
</msgitem>
<msgitem class="uresp">
<p>Search high and low for the data set.</p>
</msgitem>
</msg>
<msg>
<msgtext>This message has no number</msgtext>
<msgitem class="xpl">
<p>This message has no message number; only text.
These are really insidious because it makes finding
the message very hard.</p>
</msgitem>
</msg>
</msglist>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Message and code lists” on page 38.

Contexts

Children: text (#pcdata), APL, Bin, Char, Dec, Hex, L, Lines, MV, Num, Oct, Ph, PK, PV, SynPh, Term, TM.

Parents: Msg.

MV (message variable)

Purpose

The MV element identifies data that is a placeholder for variable data in a message. It can be used either in the message text itself or in an explanation of the message.

Examples

```
<msglist>
<msg>
<msgnum>DJI7832E</msgnum>
<msgtext>This message is issued when no data set of
the name <mv>file-name</mv> is found.</msgtext>
<msgitem class="xpl">
<p>The processor could not locate the data set named <mv>
file-name</mv>.</p>
</msgitem>
<msgitem class="severity">
<p>8</p>
</msgitem>
<msgitem class="probd">
<p>You would appear to have a problem.</p>
```

```
</msgitem>
<msgitem class="uresp">
<p>Search high and low for the data set.</p>
</msgitem>
</msg>
<msg>
<msgtext>This message has no number</msgtext>
<msgitem class="xpl">
<p>This message has no message number; only text.
These are really insidious because it makes finding
the message very hard.</p>
</msgitem>
</msg>
</msglist>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Message and code lists” on page 38.

Contexts

Children: text (#pcdata).

Parents: AnnotBody, Attention, Bridge, Caution, CompCmt, Danger, Defn, Desc, entry, Fn, L, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, MsgText, NoteBody, P, Ph, Q, SynNote, Term, Warning, Xmp, XPh.

Name (person’s name)

Purpose

The Name element must always contains a person’s name. It should not contain the name of a company (use the Company element for those).

Examples

```
<authors>
<author><person>
<name>Fred Mertz</name>
<address>
127 East Main Street,
East Overshoe, SD <postalcode>59134</postalcode>
</address>
</person></author>
</authors>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Author and Address” on page 89.

Contexts

Children: text (#pcdata), Ph.

Parents: Person.

NameLoc (named location)

Purpose

The NameLoc element associates a local ID (defined within the same document) with other locations. These other locations may be IDs within the same document, other documents, or other entities. These are referenced using named list (NMList) elements.

NameLoc is the standard HyTime mechanism for creating indirect links. NameLoc creates indirect links and can be used with all links. The indirection provided by NameLoc:

- Associates a single ID with several objects (either other SGML elements or entities).
- Associates a local ID with objects in another document or subdocument.

NameLoc is required for linking to multiple locations.

Usually NameLoc contains a single NMList element that creates a simple cross-document or multiple-object link. However, NameLoc can contain several NMList elements. For example, you can create a link to several targets in different documents by using a new NMList for the targets in each different document.

Examples

The first example shows a simple NameLoc that creates a cross-document link to a heading with an ID of `nameloc`. Note that the entity named on the DOCNAME attribute must be declared as a data entity in the document's SGML prolog.

```
   ⋮
<!ENTITY iddocref SYSTEM iddugref.idd NDATA SGMLDoc >
   ⋮
<LDESCS>
<NAMELOC ID="nmlocref">
<NMLIST DOCNAME="iddocref">nameloc</NMLIST>
</NAMELOC>
</LDESCS>
   ⋮
<P>The <L LINKEND="nmlocref">NameLoc</L> element is
the workhorse of the HyTime architecture.
```

In this example, the NameLoc element contains a single NMList element. The NMList element associates the target ID (in this case the ID of the heading reference entry for the NameLoc tag) with the document it is in (the document represented by the entity named *iddocref*). The L element itself points to the NameLoc element, which then serves to locate the actual target, the element with the ID `nameloc` in the document *iddocref*.

Attributes

ID=*nameloc_ID*
: Specifies the local ID of the NameLoc element, for the location referenced by the NameLoc element.

OBJTYPE=*target_type*
: Specifies the type of object being referenced.

This attribute must contain one of the following type names.

HEAD
A heading, division, or equivalent (such as a preface or a language element name). If the OBJTYPE attribute is omitted, HEAD is assumed.

FIG | TABLE | QUES | ANS
A figure, table, question, or answer.

STEP | CI | LI | SPOT
A step, component item, list item, or spot.

PROGRAM | ANIMATION | VIDEO | AUDIO | GRAPHIC | IMAGE
Information that is not in a document. Access to these types of information depend on the capabilities of the user's installation.

NMList
Specifies an NMList element that contains the IDs or entity names of the location being referred to by the NameLoc element.

Usage

See Chapter 12, "All about linking" on page 129.

Contexts

Children: nmlist.

Parents: LDescs.

NItem (notice item)

Purpose

The NItem element contains one or more elements that contain special notice information.

Examples

```
 ⋮
<FRONTM>
 <NOTICES>
  <NITEM><PBLK STYLE='LBLBOX'>
   <TITLE>TAKE NOTE!</TITLE>
   <P>BEFORE USING THIS INFORMATION AND THE PRODUCT IT
      SUPPORTS, BE SURE TO READ THE GENERAL INFORMATION UNDER <XREF
      REFID="NOTICES">.</P>
 </NOTICES>
 <EDNOTICES>FIFTH EDITION (AUGUST 1992)
  <P>This edition applies to Release 4 of IBM ..
  .</P>
  <P>Order publications through your IBM ...</P>
  <P>A form for reader's comments is provided ...
  </P>
 </EDNOTICES>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See "Notices and Edition notices" on page 98.

Contexts

Children: DL, Fig, L, MMObj, Note, NoteList, OL, P, PBlk, Table, UL.

Parents: Notices.

NMList (named list of IDs or entities)

Purpose

The NMList element is used within a NameLoc element to associate the NameLoc element's ID with the IDs of other elements or the IDs of other elements in other documents.

NMList also uses the DOCNAME attribute to specify the document or subdocument where the IDs are located.

NMList is used within NameLoc to contain the names (IDs or entity names) of the objects to be located, either SGML elements or entities. When the objects to be located are in the same document as the NMList element, NMList contains a list of their IDs or entity names. When the objects to be located are in another document or subdocument, specify the DocName attribute to indicate the document that defines or declares the target.

You can use several NMList elements within a single NameLoc. If objects are in the same document as the NMList element and in other documents, you need to specify a new NMList for each different document that defines or declares the target.

Examples

The first example shows a simple NMList used to create a cross-document link. Note that the entity named on the DOCNAME attribute has been declared as a data entity in the document's SGML prolog.

```
   ⋮
<!ENTITY  iddocref   PUBLIC
     "+//ISBN 0-933186::IBM//NONSGML IBMIDDoc User's Guide and
     Reference//EN" NDATA SGMLDoc >
   ⋮
<LDESCS>
  <NAMELOC ID="nmlocref">
   <NMLIST DOCNAME="iddocref">iddoc</NMLIST>
  </NAMELOC>
</LDESCS>
   ⋮
<P>The <L LINKEND="nmlocref">NameLoc</L> element is....
```

In this example, the NameLoc element, which is within an LDescs element, contains a single NMList element. The NameLoc element associates the target ID (in this case the ID of the reference entry for NMList) with the document it is in (the document represented by the entity named "iddocref"). The L element itself points to the NameLoc element, which then serves to locate the actual target, the element with the ID "iddoc" in the document "iddocref".

Attributes

NAMETYPE=<u>ELEMENT</u> | ENTITY
Indicates the contents of the NMList as follows:

ENTITY
Indicates that the names listed in the content of the NMList element are declared entity names.

ELEMENT
Indicates that the names listed in the content of the NMList element are element IDs in a document.

DOCNAME=*entity_name*
Specifies the name of the document or subdocument entity that contains the IDs or declares the entities listed in the content of the NMList element. The entity named must be declared in the current document.

If DOCNAME is not specified, the IDs in the NMList element are valid in the current document. If DOCNAME is specified, the IDs in the NMList element are valid for the document named on the DOCNAME attribute.

Note: The DOCNAME attribute is the HyTime *docorsub* attribute.

element_id
Specifies one or more element IDs of the elements to be located. When the unified name space option is in effect (either because the HyTime *unmspace* attribute was specified as Unified on the document element or because the Nametype attribute value is Unified), the IDs can also be the names of entities to be located.

DTDORLPD
Names the DTD that defines the element types or entity names used in the list. This should almost never be used. Used in conjunction with DOCNAME, it identifies a parsing context for interpreting the names in the list. The defaults will handle the vast majority of cases.

OBNAMES=OBNAMES | NOOBNAMES
OBNAMES means that referencing one location actually is an indirect reference to another location.

NOOBNAMES means that the object pointed to is the actual content of the NameLoc element that is the target.

element_id
Specifies one or more element IDs of the elements to be located.

Future Enhancement

When the unified name space option is in effect (either because the HyTime *unmspace* attribute was specified as Unified on the document element or because the NAMETYPE attribute value is Unified), the IDs can also be the names of entities to be located.

CDATA
Contains character data.

Usage

See Chapter 12, “All about linking” on page 129.

Contexts

Children: text (#pcdata).

Parents: nameloc.

Note

Purpose

The Note element contains a information that is differentiated from the main text. Information contained in a note often further explains the meaning of the main text. Use Note to create a note of one or more paragraphs.

Examples

```
<note><notebody>Thinking of a seashore, green meadow,
or cool mountain overlook can help you to relax and
be more patient.</notebody></note>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See "Notes" on page 45.

Contexts

Children: NoteBody, Title.

Parents: AnnotBody, Bridge, Cond, Copyr, DBody, Defn, Desc, DIntro, DSum, EdNotices, entry, Fig, FigSeg, Fn, LEDesc, LEDI, LI, LQ, MkNote, ModDesc, ModItem, MsgItem, NItem, Notices, P, PBlk, ProcEntry, ProcExit, ProcIntro, Safety, Sem, SynNote, TextAlt.

NoteBody (note body)

Purpose

The NoteBody element contains the body of the Note information that is differentiated from the main text.

Examples

```
<note><notebody>Thinking of a seashore, green meadow,
or cool mountain overlook can help you to relax and
be more patient.</notebody></note>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See "Notes" on page 45.

Contexts

Children: text (#pcdata), Address, Annot, APL, Bin, Bridge, CGraphic, Char, Cit, Date, Dec, DL, Fig, Formula, GL, Hex, L, Lines, LitData, LQ, MD, MkNote, MMObj, ModInfo, MV, Num, Oct, OL, P, ParmL, PBlk, Ph, PK, PV, Q, RefKey, Screen, SynPh, Syntax, Table, Term, TM, UL, Xmp, XPh, XRef.

Parents: Note.

NoteList (ordered note list)

Purpose

The NoteList element contains an ordered list of notes.

Examples

```
<notelist>
<li>Make a To Do list</li>
<li>Prioritize sensibly</li>
<li>Avoid interruptions where possible</li>
<li>Check on your progress toward monthly goals</li>
<li>Plan for the next work week</li>
<li>Do something for the fun of it</li>
<li>Spend some quality time with your pet</li>
</notelist>
```

Attributes

LINESPACE=SPACE | COMPACT
Specifies whether the items in the list should be compacted or have space between the items. Nested lists automatically inherit the setting of the outer list, but can override that default with their own setting.

| **DEF=***definition-name*
| Specifies the definition to be used. This points to a DefName attribute on a
| corresponding DEF tag. The settings defined on that DEF tag are used as
| defaults for this tag. See Chapter 9, “Using definition tags” on page 105 for
| more information.

See “Common Element Attributes (large set)” on page 227.

Usage

See “Note lists” on page 46.

Contexts

Children: Bridge, LI, LIBlk, Title.

Parents: AnnotBody, Bridge, Cond, Copyr, DBody, Defn, Desc, DIntro, DSum, EdNotices, entry, Fig, FigSeg, Fn, LEDesc, LEDI, LI, LQ, MkNote, ModDesc, ModItem, MsgItem, NItem, Notices, P, PBlk, ProcEntry, ProcExit, ProcIntro, Safety, Sem, SynNote.

Notices (contains notices)

Purpose

The Notices element contains one or more NItem elements that contain special notice information.

Examples

```
<notices><pblk style="lblbox"><title>Note</title>
<p>Before using this information, be sure to read
the general information under <xref refid="notices">.
</p>
<p>This manual was produced using IBMIDDoc SGML, the
```

```
Epic editor, and processed for print and online using
the ID Workbench.</p>
</pblk></notices>
<ednotices>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Notices and Edition notices” on page 98.

Contexts

Children: DL, Fig, L, MMObj, NItem, Note, NoteList, OL, P, PBlk, Table, UL.

Parents: FrontM.

Notloc (notation-specific location)

Purpose

The Notloc element contains a specification of an address that uses a specific notation. It enables the use of definition of anchors in non-SGML objects using specifications that are peculiar to those objects.

In the current level of IBMIDDoc, NotLoc is used to encode URLs.

The Notloc element is a HyTime element.

Use Notloc to define link anchors in non-SGML objects such as images and vector graphics, or using query specifications in a specific notation. For example, it can contain x and y pixel locations within a bitmap or the label of a graphic object in a CAD drawing. A link anchor can be the start of a link or the target of a link or both. How a Notloc-defined anchor is used and expressed depends on your online presentation system and is not defined by IBMIDDoc.

Examples

```
<IBMIDDOC>
 <PROLOG>

   ⋮
   <LDESCS>
    <NOTLOC ID="ibmwww" notation="url">
     http://www.ibm.com
    </NOTLOC>
   </LDESCS>

   ⋮
  <BODY>

   ⋮
   <P PROPS="www">Be sure to check out the
   <L LINKEND="ibmwww"> for the latest IBM product information.
   </P>
```

Notloc (notation-specific location)

Attributes

ID=*notloc_id*
: Contains the ID of this Notloc element.

NOTATION=*notation_name*
: Specifies the notation of the address specification. It must be a declared SGML notation.

Usage

See Chapter 12, “All about linking” on page 129.

Contexts

Children: text (#pcdata).

Parents: LDescs.

Num (number)

Purpose

Use the Num element to identify numbers in a base for which a more precise element, such as Hex or Bin, does not exist. You must specify the Base attribute to indicate the base of the number, for example, "36" for base 36 numbers.

Examples

```
<P>Nums in base 34 use the digits zero (0) to nine (9)
and the letters A to Z minus I and O (or L and O), for
example, <NUM BASE="34">Z</NUM> = <DEC>33</DEC>.
```

Attributes

BASE=*basevalue*
: Contains an integer value specifying the base of the number.

See “Common Element Attributes (large set)” on page 227.

Usage

See “Highlighting” on page 43.

Contexts

Children: text (#pcdata).

Parents: AnnotBody, Attention, Bridge, Caution, CompCmt, Danger, Defn, Desc, entry, Fn, L, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, MsgText, NoteBody, P, Ph, Q, SynNote, Term, Warning.

ObjLib (object library)

Purpose

The ObjLib element contains elements that are to be used elsewhere in the document by reference. You can have as many different ObjLib elements as desired. Use object libraries to collect elements for reuse. Elements in object libraries are not processed at the place they occur in the source document but are processed only when referred to by another element of the same type using the CONLOC or RETALTS attributes. The ObjLib needs to be specified in the document’s Prolog.

Use the optional Desc element to describe the purpose of the library.

Migration Note

ObjLib replaces the function provided by DVCF side files for organizing collections of text for re-use.

Examples

```
<OBJLIB>
 <OBJLIBBODY>
  <DLENTRY ID="FILEMENUITEM" "CLASS"="menuitem">
   <TERM>File</TERM>
   <DEFN>Work with files.</DEFN>
  </DLENTRY>
  <DLENTRY ID="Editmenuitem" CLASS="menuitem">
   <TERM>Edit</TERM>
   <DEFN>Perform edit functions.</DEFN>
  </DLENTRY>
 </OBJLIBBODY>
</OBJLIB>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Reusing elements from an object library” on page 191.

Contexts

Children: Desc, ObjLibBody.

Parents: DProlog, Prolog, SpecDProlog.

ObjLibBody (object library body)

Purpose

The ObjLibBody element contains the elements in an object library.

The OBJLibBody contains all of the elements in the object library.

Examples

```
<OBJLIB>
 <OBJLIBBODY>
  <DLENTRY ID="FILEMENUITEM" "CLASS"="menuitem">
   <TERM>File</TERM>
   <DEFN>Work with files.<DEFN>
  </DLENTRY>
  <DLENTRY ID="Editmenuitem" CLASS="menuitem">
   <TERM>Edit</TERM>
```

```
    <DEFN>Perform edit functions.</DEFN>
  </DLENTRY>
 </OBJLIBBODY>
</OBJLIB>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See "Reusing elements from an object library" on page 191.

Contexts

Children: any element.

Parents: ObjLib.

ObjRef (object reference)

Purpose

The ObjRef element references a declared graphic entity. ObjRef has no content; it only references the graphic entity using the entity's declared name.

When using an SGML editor, such as ArborText's Epic*Editor, that supports the inline display of certain graphic types, the graphic entity will be displayed inline with your text.

Examples

```
<!ENTITY bike system "bike.gif" ndata graphics>
  ...
<MMOBJ>
<OBJREF OBJ="bike">
<TEXTALT>This is a two-wheeled bicycle.</TEXTALT>
</MMOBJ>
```

Attributes

OBJ=*entity_name*
This attribute's value is the name of the graphic entity containing the graphic to be included.

EDITSCALE=*scaling_value*
The EDITSCALE attribute specifies the scaling factor for graphics in an SGML editor capable of displaying graphics inline.

WIDTH and DEPTH
WIDTH and DEPTH attributes allow you to expand or contract a drawing. They accept the following values:

*x.y***in**
inches. For example: `4in` or `5.5in`.

*x.y***pt**
points. For example: `48pt` or `39.5pt`.

*x.y***pi**
picas. y is base 12. For example, 5.11pi is 5 picas, 11 points.

*x.y***mm**
: millimeters. For example: `55mm` or `47.6mm`.

*x.y***cm**
: centimeters. For example: `12cm` or `10.7cm`.

ScaleBox=BestFit | DepthFirst | UseBoth | None
Defines how to use the width and depth values to scale an object.

BestFit
Scaled to fit specfied area without skewing the artwork. This is the default value.

DepthFirst
The depth value will be chosen first it the composer must skew the object.

UseBoth
Sets to the depth and width values specified. If any one value is specified, it will be used without without skewing.

None
No scaling.

Usage

See "Including artwork in documents" on page 55.

Contexts

Children: empty.

Parents: MMObj.

Oct (octal number)

Purpose

Use the Oct element to identify octal data, which is encoded in a base-8 numbering system.

Examples

```
<BIN>11000001</BIN> = <DECIMAL>193</DECIMAL> = <OCT>301</OCT>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See "Highlighting" on page 43.

Contexts

Children: text (#pcdata).

Parents: AnnotBody, Attention, Bridge, Caution, CompCmt, Danger, Defn, Desc, entry, Fn, L, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, MsgText, NoteBody, P, Ph, Q, SynNote, Term, Warning.

OL (ordered list)

Purpose

The OL element contains a list of items where the order of the items has particular significance, or where the ordinal number of each item is significant.

Examples

```
<ol>
<li>Cream butter and sugar together until fluffy.</li>
<li>Beat in egg yolks one at a time.</li>
<li>Add nutmeg, cinnamon, and vanilla, and mix thoroughly.
The batter should be smooth and glossy and stream
off the spoon in ribbons.</li>
<li>Fold in beaten egg whites.
<p>Do not overmix; the batter should be light and fluffy.</p></li>
</ol>
```

Attributes

LINESPACE=SPACE | COMPACT
Specifies whether the items in the list should be compacted or have space between the items. Nested lists automatically inherit the setting of the outer list, but can override that default with their own setting.

OLType
This attribute specifies the list type. Allowed values are:

Normal
Causes a normal, ordered list.

Step Causes a list with the word ″Step″ to appear before the step item.

Checkoff
Causes a small check-off area to appear before the step item.

CheckoffStep
Causes both a small check-off area and the word ″Step″ to appear before the step item.

| **DBScalePct=***scale-percent*
| You can use the DBScalePct attribute to scale the dingbat (the thing in front of
| the list) up or down for a list item. The scale-percent is one of the following
| numbers: 50, 60, 70, 80, 90, 100, 110, 120, 140, 160, 180, 200, 240, or 300. 100 is
| the normal size; 50 is 50% smaller; 200 is 200% larger. For example, this scales
| the bullet or number to twice its size:

```
dbscalepct="200"
```

| This works for hardcopy only.

| **DEF=***definition-name*
| Specifies the definition to be used. This points to a DefName attribute on a
| corresponding DEF tag. The settings defined on that DEF tag are used as
| defaults for this tag. See Chapter 9, “Using definition tags” on page 105 for
| more information.

Seq
Specifies that a sequence of ordered lists are to connect. That is, the list can end, but when the list starts again, the numnbering continues from the previous list’s last item. Use this for steps that need to cross divisions or table

cells. The value START indicates the beginning of the list; the value END indicates the end of the list. The ID on the list must appear on the SEQID attributes on the continuing lists.

SEQID
Indicates the list is part of a sequence. The SEQID points to the ID of the beginning list.

See "Common Element Attributes (large set)" on page 227.

Usage

See "Ordered lists" on page 30.

Contexts

Children: Bridge, LI, LIBlk.

Parents: AnnotBody, Attention, BackCover, Bridge, Caution, Cond, Copyr, Danger, DBody, Defn, Desc, DIntro, DSum, EdNotices, entry, Fig, FigSeg, Fn, FrontCover, LEDesc, LEDI, LI, LQ, MkNote, ModDesc, ModItem, MsgItem, NItem, NoteBody, Notices, P, PBlk, ProcEntry, ProcExit, ProcIntro, Safety, Sem, SynNote, TextAlt, Warning.

OLDef (Ordered list definition)

Purpose

The OLDef element sets attribute defaults for ordered lists and note lists. OLDef goes within the document prolog to set definitions for the entire document; or within a division prolog to set definitions for just that division. The OLDef tag goes inside a PropDefs tag.

Examples

```
<propdefs>
<oldef defname="setup" linespace="space" oltype="checkoffstep" dbscalepct="140">
</propdefs>
...
<ol def="setup">
<li>Open the box.</li>
<li>Jump out</li>
<li>Enjoy the out-of-the-box experience!</li>
</ol>
```

Attributes

DEFNAME=*definition-name*
The DefName attribute identifies a definition element. DefName values must be unique within a single document. A element that has a DEF attribute can point a corresponding tag with a DefName attribute. DefName values can be up to 64 characters long. DefName values must start with an alphabetic character and can contain letters, numbers, dashes (-), or periods (.).

Definition tags can be specified without a DefName attribute in the document's prolog. This allows you to change the initial settings of attributes for the entire document when the DEF tag is in the document's prolog. In a division's prolog, it changes the initial settings for that division.

DBScalePct=*scale-percent*
: You can use the DBScalePct attribute to scale the dingbat (the thing in front of the list) up or down for a list item. The scale-percent is one of the following numbers: 50, 60, 70, 80, 90, 100, 110, 120, 140, 160, 180, 200, 240, or 300. 100 is the normal size; 50 is 50% smaller; 200 is 200% larger. For example, this scales the bullet or number to twice its size:

```
dbscalepct="200"
```

This works for hardcopy only.

LINESPACE=SPACE | COMPACT
: Specifies whether the items in the list should be compacted or have space between the items. Nested lists automatically inherit the setting of the outer list, but can override that default with their own setting.

OLType
: This attribute specifies the list type. Allowed values are:

Normal
: Causes a normal, ordered list.

Step
: Causes a list with the word ″Step″ to appear before the step item.

Checkoff
: Causes a small check-off area to appear before the step item.

CheckoffStep
: Causes both a small check-off area and the word ″Step″ to appear before the step item.

Props=*properties*
: The Props attribute is used to specify the condition under which the information contained within the element appears. See “Property-Based Retrieval” on page 195.

Usage

See Chapter 9, “Using definition tags” on page 105.

Contexts

Children: empty.

Parents: PropDefs, PropGroup.

Oper (syntax operator)

Purpose

Use Oper to define an operator within a syntax definition. Operators usually connect two parts of a statement and imply an action such as assignment or comparison. Operators can also be applied to a single parameter, such as the negation operator. Typical operators are the equals sign (=) and the mathematical operators such as add (+) and multiply (*).

Examples

```
<syntax>
<group>
<kwd>LANGUAGE</kwd>
```

```
<oper>=</oper>
<var>language_name</var>
</group>
</syntax>
```

Attributes

OPTREQ= OPT|REQ|DEF
Indicates whether the operator is optional, required, or the default.

LINKEND=*id*
Contains an ID reference that enables a link to an arbitrary location in the document.

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 15, “Programming Syntax Diagrams” on page 147.

Contexts

Children: text (#pcdata).

Parents: Group, SynPh.

OrderNum (order number)

Purpose

The OrderNum element contains the order number assigned to a document. This element is for use by non-IBM documents. IBM documents use the IBMDocNum element (see “IBMDocNum (IBM document number)” on page 298).

Examples

```
<ORDERNUM>GC12-3456-00</ORDERNUM>
```

Attributes

#PCDATA
Contains the order number.

Contexts

Children: text (#pcdata), Ph.

Parents: BibEntry, LibEntry.

OrigIBMDocNum (original IBM document number)

Purpose

Use this when you have a new manual that superceeds a previous manual.

Contexts

Children: text (#pcdata), Ph.

Parents: IBMBibEntry.

Owners

Purpose

Contains the name of the owner or owners of the information.

Examples

```
<OWNERS>
 <PERSON>
  <NAME>John Smith</NAME>
  <ADDRESS>XYZ Corp. RTP, NC  27709</ADDRESS>
 </PERSON>
 <PERSON>
  <NAME>Susan Jones</NAME>
  <ADDRESS>ABC Corp. RTP, NC</ADDRESS>
 </PERSON>
</OWNERS>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Contexts

Children: Corp, Person.

Parents: DProlog, Prolog, SpecDProlog.

P (paragraph)

Purpose

The P element contains a paragraph; a block of text representing a single idea. Use paragraphs to contain flowing text and other elements associated with the text, such as lists, examples, figures, and tables.

Paragraphs with no content are not valid.

Examples

This example shows a simple paragraph:

```
<P>This is a simple paragraph.</P>
```

Use the paragraph end tag to control whether or not other elements are contained within the paragraph. In this example, the unordered list is contained within the paragraph:

```
<P>This paragraph contains a list:
 <UL>
  <LI>List item</LI>
  <LI>List item</LI>
 </UL>
</P>
```

To keep the list from being part of the paragraph, the paragraph must be ended before the list begins, as in this example:

```
<P>This paragraph does not contain the list.</P>
<UL>
 <LI>List item</LI>
</UL>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Creating paragraphs (P element)” on page 21.

Contexts

Children: text (#pcdata), Address, Annot, APL, Bin, Bridge, CGraphic, Char, Cit, Date, Dec, DL, Fig, Formula, GL, Hex, L, Lines, LitData, LQ, MD, MkNote, MMObj, ModInfo, MV, Note, NoteList, Num, Oct, OL, ParmL, Ph, PK, PV, Q, RefKey, Screen, SynPh, Syntax, Table, Term, TM, UL, Xmp, XPh, XRef.

Parents: AnnotBody, Attention, BackCover, Bridge, Caution, Cond, Copyr, Danger, DBody, Defn, Desc, DIntro, DSum, EdNotices, entry, Fig, FigSeg, Fn, FrontCover, LEDesc, LEDI, LI, LQ, MkNote, ModDesc, ModItem, MsgItem, NItem, NoteBody, Notices, PBlk, ProcEntry, ProcExit, ProcIntro, Safety, Sem, SynNote, TextAlt, Warning.

Parm (parameter list entry)

Purpose

The Parm element contains a single parameter and its definition within a parameter list.

Examples

```
<parml>
<parm><term>KEYWORD = <pk optreq="DEF">DEFAULT</pk>|VALUE
</term>
<defn>This is the description of the parameter above.
It could go on for many pages, if necessary. (Of course,
that means we have a very complicated parameter to
describe.)</defn>
</parm>
<parm><term>KEYWORD2 = &lbrc;ABC|XYZ&rbrc;</term>
<term>&lbrk;KEYWORD3 = GGG&rbrk;</term>
<defn>This description applies to the two parameters
above. Often in examples of programming syntax, it
is necessary to use symbols for the brackets and braces.
</defn>
</parm>
<parm><term><synph><kwd>KEYWORD3</kwd></synph></term>
<defn>Here's a term that uses the syntax phrase (SYNPH);
it allows you to use the same items as a syntax diagram.
</defn>
</parm>
</parml>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Parameter lists” on page 34.

Parm (parameter list entry)

Contexts

Children: Defn, Term.

Parents: ParmBlk, ParmL.

ParmBlk (parameter list block)

Purpose

The ParmBlk element organizes parameter list entries into meaningful groupings. For example, if you group parameters within the definition of a complex statement or command, you can use ParmBlk in the parameter list to mirror the grouping in the syntax definition.

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Grouping list items” on page 37.

Contexts

Children: Bridge, Parm, Title.

Parents: ParmL.

Examples

```
<parml>
<parmblk>
<parm><term>one term</term>
<defn>definition</defn>
</parm>
<parm><term>another term</term>
<defn>definition</defn>
</parm>
</parmblk>
<parm><term>yet another term</term>
<defn>definition</defn>
</parm>
</parml>
```

ParmL (parameter list)

Purpose

Use parameter lists to describe the parameters in a computer language statement. Parameter lists are usually associated with syntax definitions. Entries can be organized within parameter lists using ParmBlk elements. Bridge elements can also be used to create connections between blocks of entries, including syntax definitions.

Examples

```
<parml>
<parm><term>KEYWORD = <pk optreq="DEF">DEFAULT</pk>|VALUE
</term>
<defn>This is the description of the parameter above.
```

```
It could go on for many pages, if necessary. (Of course,
that means we have a very complicated parameter to
describe.)</defn>
</parm>
<parm><term>KEYWORD2 = &lbrc;ABC|XYZ&rbrc;</term>
<term>&lbrk;KEYWORD3 = GGG&rbrk;</term>
<defn>This description applies to the two parameters
above. Often in examples of programming syntax, it
is necessary to use symbols for the brackets and braces.
</defn>
</parm>
<parm><term><synph><kwd>KEYWORD3</kwd></synph></term>
<defn>Here's a term that uses the syntax phrase (SYNPH);
it allows you to use the same items as a syntax diagram.
</defn>
</parm>
</parml>
```

Attributes

TERMWIDTH= SMALL | MEDIUM | LARGE | 1 | 2
You can use the TERMWIDTH attribute to determine the indentation size of the definition list. The valid choices are: small (.25 inch, the default), medium (.5 inch), and large (1 inch). The value “1” is for 1-character width; “2” is for a 2-character width.

LINESPACE=SPACE | COMPACT
Specifies whether the items in the list should be compacted or have space between the items. Nested lists automatically inherit the setting of the outer list, but can override that default with their own setting.

| **DEF=***definition-name*
| Specifies the definition to be used. This points to a DefName attribute on a
| corresponding DEF tag. The settings defined on that DEF tag are used as
| defaults for this tag. See Chapter 9, “Using definition tags” on page 105 for
| more information.

| **HeadStyle=base_h | italic_h | bold_h | monospaced_h | underlined_h |**
| **bold_italic_h | italic_underlined_h | bold_underlined_h |**
| **bold_italic_underlined_h**
| Specifies the highlighting to use for the list’s TermHd and DefnHd heading
| tags. The default is bold.

| All values work for Xyvision, XHTML, HTML, and BookMaster. For IPF or
| RTF, there are some inconsistencies due to the limitations of those formats:
| “monospaced” is ignored, and the default style (bold) is used for terms and
| headings. “bold_italic_underlined” does not exist in IPF or RTF; this is treated
| the same as bold_underlined. “base” works on everything but definition terms;
| there is no plain style in IPF, so the default bold will not be over-ridden.

| **TermStyle=base | italic | bold | monospaced | underlined | bold_italic |**
| **italic_underlined | bold_underlined | bold_italic_underlined**
| Specifies the highlighting to use for the list’s Term tags. The default is bold.

See “Common Element Attributes (large set)” on page 227.

Usage

See “Parameter lists” on page 34.

Contexts

Children: Bridge, DefnHd, Parm, ParmBlk, TermHd.

Parents: AnnotBody, Attention, Bridge, Caution, Danger, DBody, Defn, DIntro, DSum, entry, Fig, FigSeg, Fn, LEDI, LI, LQ, MkNote, ModDesc, ModItem, MsgItem, NoteBody, P, PBlk, ProcIntro, SynNote, Warning.

Part (major document part)

Purpose

Use Part to divide a document's chapters into logical groupings. For example, in a document that contains both guide and reference information, you can define two parts, one containing the guide information and the other containing the reference information.

The Part element does not change the logical hierarchy of the divisions it contains. For example, if, in your document style, first-level divisions are considered to be chapters, they are still chapters when contained within Part. Thus, the enumeration of divisions contained within Parts is not affected by the presence or absence of Part elements.

Examples

```
<ibmiddoc>
<body>
<part>
<dprolog><titleblk>
<title>Introduction</title>
</titleblk></dprolog>
<dbody>
<d>
<dprolog><titleblk>
<title>Salads of our neighborhood</title>
</titleblk></dprolog>
<dbody></dbody></d>
<d>
<dprolog><titleblk>
<title>Salads of the world</title>
</titleblk></dprolog>
<dbody></dbody></d>
</dbody></part>
<part>
<dprolog><titleblk>
<title>Recipies</title>
</titleblk></dprolog>
<dbody>
<d>
<dprolog><titleblk>
<title>Egg salad</title>
</titleblk></dprolog>
<dbody></dbody></d>
<d>
<dprolog><titleblk>
<title>Tuna fish salad</title>
</titleblk></dprolog>
<dbody></dbody></d>
</dbody></part></body>
</ibmiddoc>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See "Using parts to organize your chapters" on page 24.

Contexts

Children: Abstract, DBody, DIntro, DProlog, DSum.

Parents: Body.

PartAsm (part assembly)

Purpose

The PartAsm element contains the elements needed to construct a parts assemby list.

Examples

```
<partasm id="bike" style="bkm:(layout=same)"><title>
Bicycle</title><mmobj><objref obj="bike">
<textalt>Bicycle</textalt>
</mmobj><compl>
<ci idxnum="1" partnum="4563423" upa="1">Bike</ci>
<compl>
<ci idxnum="1" partnum="1230987" upa="1">Frame</ci>
<ci idxnum="2" partnum="1238475" upa="1">Wheel assembly, front</ci>
<compcmt>For detailed breakdown, see <xref refid="wheelxmp">.</compcmt>
<ci idxnum="3" partnum="1234939" upa="1">Wheel assembly, rear</ci>
</compl>
</compl>
</partasm>
```

Attributes

Toc=toc | notoc
Specifies whether the heading should appear in the table of contents. The default is for headings to appear in the table of contents; to a certain level (such as heading level 3). This replaces the older style=hidden attribute. Use this to prevent headings from appearing in the table of contents. It cannot be used to add a heading to the table of contents that would not normally appear. The levels of headings that appear in the table of contents are determined by the MaxToc attribute of the IBMIDDoc tag.

See "Common Element Attributes (large set)" on page 227.

Usage

See Chapter 23, "Creating parts catalog lists" on page 215.

Contexts

Children: CompL, MMObj, PartAsmSeg, RetKey, Title.

Parents: Appendix, Body, DBody, LEDI, ModItem, MsgItem.

PartAsmSeg (part assembly segment)

Purpose

The PartAsmSeg element contains the component elements needed to contain the parts assembly information that is a logical division of the assembly.

Attributes

See “Common Element Attributes (large set)” on page 227.

Contexts

Children: CompL, MMObj.

Parents: PartAsm.

PBlk (paragraph block)

Purpose

Use PBlk to group paragraphs and paragraph-like elements together. You can use a Title element on the PBlk element to identify or introduce the topic that the paragraphs in the PBlk address. PBlk can be used to define property values for the set of contained elements. For example, if a number of paragraphs have changed, you can put them within a PBlk element in order to define their revision status.

You can also use PBlk within an ObjLib to define groups of paragraphs for use by reference. For example, if you have a figure with an introductory paragraph that you want to use in several contexts, you can put the paragraph and the figure into a PBlk. To use the content of that PBlk in several places, you can specify a PBlk with a CONLOC attribute that refers to the PBlk you want to reuse.

PBlk also enables you to define a set of paragraphs as a single link anchor by linking to the PBlk element.

To create a labeled box, use attribute `style="lblbox"` on the PBlk tag:

```
<pblk style="lblbox"><title>Getting There</title>
<p>To get to...
</p></pblk>
```

To create hidden text in IPF and Windows, use attribute `style="hidden"` on the PBlk tag:

Attributes

style=hidden
A PBLK with style=hidden and formated for IPF or RTF becomes a hidden division. Be sure to specify a title for the PBLK or you will get *** for the title of the generated division.

```
<pblk style="hidden"><title>Getting There</title>
<p>To get to...
</p></pblk>
```

style=lblbox
This causes a labeled box to surround the content. Use a Title tag to specify the title for the labeled box.

```
<pblk style="lblbox"><title>Getting There</title>
<p>To get to...
</p></pblk>
```

Contexts

Children: Annot, AsmList, Attention, BibList, Bridge, Caution, CGraphic, Danger, DL, Fig, FnList, GL, L, Lines, LitData, LQ, MarkList, MkNote, MMObj, ModInfo, Note, NoteList, OL, P, ParmL, PBlk, Screen, Syntax, Table, Title, UL, Xmp.

Parents: AnnotBody, Attention, Bridge, Caution, Cond, Copyr, Danger, DBody, Defn, Desc, DIntro, DSum, EdNotices, entry, Fig, FigSeg, Fn, FrontCover, LEDesc, LEDI, LI, LQ, MkNote, ModDesc, ModItem, MsgItem, NItem, NoteBody, Notices, PBlk, ProcEntry, ProcExit, ProcIntro, Safety, Sem, SynNote, TextAlt, Warning.

Person (person's name and address)

Purpose

The Person element contains name and address pairs for use in Author, Approvers, and Owners where either a person or an enterprise can be meaningful.

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See "Author and Address" on page 89.

Contexts

Children: Address, Name.

Parents: Approvers, Author, Maintainer, Owners.

Examples

```
<owners><person><name>Mike Temple</name></person>
</owners>
```

Ph (Phrase)

Purpose

Use the Ph element to identify a phrase for some reason not already provided for by the IBMIDDoc language. Phrases can define containment structures to associate one element with another, such as associating a footnote with a specific sentence, or they can use an author-defined element class to further specify the semantic meaning of a phrase.

The Ph element can also be used to associate a specific property with a specific phrase. For example, you can associate a revision or version level with a phrase. You can also identify a word as a particular type of data for special processing.

You can precisely identify information that is unique to your document by defining element classes with the ClassDef element and using those classes with the Ph element. For example, in the documentation for a program that supports mining operations, you may need to discuss different kinds of rocks and want to precisely identify references to different rocks to enhance the retrievability of your information. You can define element classes for the different rock types and use Ph elements with those classes to identify references to the types of rock.

Examples

Hey there! This is **very** **important**! Don't go out in the *rain* ***without your galoshes***!

Here's its markup:

```
<ph style="Bold Italic">Hey there!</ph>
This is <ph style="Underlined Bold">very</ph>
<ph style="Bold">important</ph>! Don't go out in the
<ph style="Italic">rain</ph> <ph style="Underlined Bold Italic">
without your galoshes</ph>!
```

Attributes

style=*phrase-style*
The style attribute values include:

- base
- **bold**
- *italic*
- ***bold italic***
- underlined
- $^{\text{superscript}}$
- $_{\text{subscript}}$
- `monospaced`
- SMALLCAPS. Note that YOU need to do the uppercase conversion yourself. This is because not all languages do proper uppercase conversion of lowercase letters.
- **underlined bold**
- *underlined italic*
- ***underlined bold italic***
- UNDERLINED SMALLCAPS

See "Common Element Attributes (large set)" on page 227.

Usage

See "Highlighting" on page 43.

When migrating a Bookmaster document, Bookmaster highlight phrases are migrated to IBMIDDoc phrases.

HP0	<PH STYLE="base">
HP1	<PH STYLE="italic">
HP2	<PH STYLE="bold">
HP3	<PH STYLE="bold italic">
HP4	<PH STYLE="smallcaps">
HP5	<PH STYLE="underlined">
HP6	<PH STYLE="underlined italic">

HP7	<PH STYLE=″underlined bold″>
HP8	<PH STYLE=″underlined bold italic″>
HP9	<PH STYLE=″underlined smallcaps″>

Contexts

Children: text (#pcdata), Address, APL, Bin, Char, Cit, Date, Dec, Formula, Hex, L, MD, MV, Num, Oct, Ph, PK, PV, Q, RefKey, SynPh, Term, TM, XPh, XRef.

Parents: Address, AnnotBody, Attention, BOFNum, Bridge, Cap, Caution, CGraphic, CI, CLE, Code, CompCmt, Cond, Copyr, Danger, Defn, DefnHd, Desc, entry, ExternalFileName, FileNum, Fn, IBMBOFNum, IBMDocNum, IBMFeatNum, IBMPartNum, IBMPgmNum, IdxTerm, ISBN, L, LEDesc, LEN, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, ModLvl, ModName, MsgNum, MsgText, Name, NoteBody, OrderNum, OrigIBMDocNum, P, Ph, ProcEntry, ProcExit, PrtLoc, PublicId, Q, Release, RetKey, Screen, Sem, STitle, SubTitle, SynNote, Term, TermHd, TextAlt, Title, TM, Version, VolId, Warning, Xmp, XPh.

Phone (telephone number)

Purpose

The Phone element contains a telephone number.

Phone has the EQUIP attribute that is used to specify the type of phone equipment being described.

If you want to specify a fax and a voice phone, use two Phone elements within the Address element.

Examples

```
<phone equip="fax">1-800-555-1212</phone>
```

Attributes

EQUIP=FAX|VOICE|VOICEFAX
Specifes the type of telephone equipment being described.

See “Common Element Attributes (large set)” on page 227.

Usage

See “Using reader’s comment form (RCF)” on page 103.

Contexts

Children: text (#pcdata).

Parents: Address.

PK (programming keyword)

Purpose

The PK element contains a programming keyword. A keyword is a literal string that has special significance in the context in which it is being used.

Examples

```
<P>Specify the user ID and password parameters when error messages
indicate that you must provide security information and specifying the
<PK>-n</PK> parameter does not solve the problem.
See <XREF REFID="SECURITY"> for an explanation of using security
parameters in the....</P>
```

Attributes

OPTREQ=OPT | REQ | DEF
Indicates if the keyword is optional, required, or the default. REQ (required) is the default value for this attribute.

Usage

See "Highlighting" on page 43.

Contexts

Children: text (#pcdata).

Parents: AnnotBody, Attention, Bridge, Caution, CompCmt, Danger, Defn, Desc, entry, Fn, L, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, MsgText, NoteBody, P, Ph, Q, SynNote, Term, Warning, Xmp, XPh.

PNIndex (part number index)

Purpose

The PNIndex element is a specialized list element that contains an index of all parts contained in CI (component item) elements.

Examples

```
  ⋮
<PNINDEX SPEC=AUTO>
  ⋮
```

Attributes

Toc=toc | notoc
Specifies whether the heading should appear in the table of contents. The default is for headings to appear in the table of contents; to a certain level (such as heading level 3). This replaces the older style=hidden attribute. Use this to prevent headings from appearing in the table of contents. It cannot be used to add a heading to the table of contents that would not normally appear. The levels of headings that appear in the table of contents are determined by the MaxToc attribute of the IBMIDDoc tag.

| **LAYOUT=Default-Layout | OneCol | TwoCol | ThreeCol**
| Specifies the column-style for this portion of the book.

| **Default-Layout**
| The section uses the default layout for the document style.

| **OneCol**
| The entries format across the entire page.

| **TwoCol**
| The entries format in two columns.

| **ThreeCol**
| The entries format in three columns.

SPEC=AUTO | MAN
This attribute has a fixed value of AUTO. The manual value currently does not work.

See "Common Element Attributes (large set)" on page 227.

Usage

See "Getting a part number index" on page 220.

Contexts

Children: GendTitle, RetKey, TitleBlk.

Parents: BackM.

PostalCode (postal or zip code)

Purpose

The PostalCode element contains a zip or postal code.

Examples

```
<authors>
<author><person>
<name>Fred Mertz</name>
<address>125 West Hollywood Blvd
Tinseltown, CA <postalcode>90210</postalcode></address>
</person></author>
</authors>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See "Using reader's comment form (RCF)" on page 103.

Contexts

Children: text (#pcdata).

Parents: Address.

Preface

Purpose

The Preface element contains introductory information about a document, such as the purpose of the document. If you wish to enter a unique title for the Preface, use the TitleBlk element to contain the Title element and the title text.

Examples

```
<FRONTM>
 <PREFACE><SPECDPROLOG><GENDTITLE></SPECDPROLOG>
  <P>This information is...

   ⋮
</FRONTM>
```

Attributes

LAYOUT=Document-Layout | OneCol | TwoCol | OffsetCol
Specifies the column-style for this portion of the book.

Document-Layout
The section uses the default layout for the document style.

OffsetCol
Formats the text using an indented, one-column layout. The headings format across the page, with the offset text, or indented from the margin.

OneCol
The text formats across the entire page.

TwoCol
The text formats in two columns.

Toc=toc | notoc
Specifies whether the heading should appear in the table of contents. The default is for headings to appear in the table of contents; to a certain level (such as heading level 3). This replaces the older style=hidden attribute. Use this to prevent headings from appearing in the table of contents. It cannot be used to add a heading to the table of contents that would not normally appear. The levels of headings that appear in the table of contents are determined by the MaxToc attribute of the IBMIDDoc tag.

See "Common Element Attributes (large set)" on page 227.

Usage

See "The preface" on page 100.

Contexts

Children: DBody, DIntro, DSum, SpecDProlog.

Parents: FrontM.

Proc (procedure)

Purpose

The Proc element structures information that describes a procedure or task. The markup enables a wide variety of print and online presentation styles. It contains a link element (RefKey) for connecting procedure descriptions to graphics and other multimedia elements, such as animation or tutorials.

Use the procedure element to describe procedures such as user tasks. A procedure consists of three basic parts: a procedure entry, one or more steps, and a procedure exit. The procedure entry defines the entry criteria for a procedure, such as any prerequisite tasks or related tasks. Each procedure step defines the actions to take and the expected results and can contain other procedures. The procedure exit describes the expected result of performing the task and what to do next.

Examples

```
<proc id="babymap" style="BKM:(STYLE=BASE SEP=INLINE COMPACT)">
<titleblk><title>Baby Johnny is Crying</title></titleblk>
<procentry>Six-month old baby Johnny was sleeping
peacefully. Suddenly he began to cry.</procentry>
<procstep>
<proccmnd>
<desc>Check Johnny's diaper.</desc>
</proccmnd>
<decisionpnt>
<cond>Is the diaper wet?</cond>
<then><procstep><proccmnd>
<desc>Change the diaper.</desc>
</proccmnd><procexit>Johnny was uncomfortable.</procexit>
</procstep>
</then>
<else>
<desc>Continue at <xref refid="hungry">.</desc>
</else>
</decisionpnt>
</procstep><procstep id="hungry">
<decisionpnt>
<cond>Is Johnny hungry?</cond>
<then><procstep><decisionpnt>
<cond>Does Johnny have teeth?</cond>
<then><procstep><stepnotes><li>Johnny can eat solid
food.</li>
<li>Continue at <xref refid="frozstk"></li>
</stepnotes></procstep>
</then>
<else><procstep id="bottle"><proccmnd>
<desc>Warm a bottle.</desc>
</proccmnd><proccmnd>
<desc>Feed Johnny.</desc>
</proccmnd><procexit>Johnny needed a bottle.</procexit>
</procstep>
</else>
</decisionpnt></procstep>
</then>
<else><procstep><proccmnd>
<desc>Rock Johnny to sleep.</desc>
</proccmnd><procexit>Johnny was sleepy.</procexit>
</procstep>
</else>
</decisionpnt>
</procstep><procstep id="frozstk">
<proccmnd>
<desc>Thaw and broil a steak for Johnny. Include a
baked potato with butter and sour cream.</desc>
</proccmnd>
<procexit>Johnny was really hungry.</procexit>
</procstep></proc>
```

Attributes

procnum=*procedure-number*
: Assigns a specific number to a procedure.

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 22, “Creating maintenance analysis procedures” on page 207.

Contexts

Children: Desc, ProcEntry, ProcExit, ProcIntro, ProcStep, ProcSumm, RetKey, TitleBlk.

Parents: Appendix, Body, DBody, Else, LEDI, ModItem, MsgItem, ProcCmnd, Then.

ProcCmnd (procedure command)

Purpose

The ProcCmnd element contains the command text for the procedure described in the procedure’s Desc element.

Use ProcCmnd to direct the user to take a specific action. More than one ProcCmnd element can be used on a ProcStep, but multiple ProcCmnd elements should be very closely related. If they are not closely related , they should be contained in separate ProcSteps.

Examples

```
<proccmnd>
<desc>Check Johnny's diaper.</desc>
</proccmnd>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 22, “Creating maintenance analysis procedures” on page 207.

Contexts

Children: Desc, Proc, ProcStep.

Parents: ProcStep.

ProcEntry (procedure entry point)

Purpose

The ProcEntry element defines the entry criteria for a given procedure and describes the procedure itself. The PREREQPROCS and RELPROCS attributes reference prerequisite or related procedures.

Examples

```
<procentry>Six-month old baby Johnny was sleeping
peacefully. Suddenly he began to cry.</procentry>
```

Attributes

PREREQPROCS
: This attribute's value references one or more prerequisite procedures. The order the procedure IDs are specified indicates the order the prerequisite procedures should be performed.

RELPROCS
: The value of this attribute references a one or more related, but not prerequisite, procedures.

See "Common Element Attributes (large set)" on page 227.

Usage

See Chapter 22, "Creating maintenance analysis procedures" on page 207.

Contexts

Children: text (#pcdata), Attention, Caution, Danger, DL, Fig, L, MMObj, Note, NoteList, OL, P, PBlk, Ph, StepRef, Table, Term, TM, UL.

Parents: Proc, ProcSummItem.

ProcExit (procedure exit point)

Purpose

The ProcExit element describes the exit criteria for a procedure and optionally contains information about what to do next and how to recover if something went wrong.

Examples

```
<procexit>Johnny needed a bottle.</procexit>
```

Attributes

RECOVERYPROC
: This attribute references a recovery procedure that describes what to do if the procedure is not completed successfully.

See "Common Element Attributes (large set)" on page 227.

Usage

See Chapter 22, "Creating maintenance analysis procedures" on page 207.

Contexts

Children: text (#pcdata), DL, Fig, L, MMObj, Note, NoteList, OL, P, PBlk, Ph, StepRef, Table, Term, TM, UL.

Parents: Proc, ProcStep, ProcSummItem.

ProcIntro (procedure introduction)

Purpose

The ProcIntro element contains the introduction to a procedure.

Examples

```
<procintro>
<p>A father's quick guide to child care.</p>
</procintro>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See Chapter 22, "Creating maintenance analysis procedures" on page 207.

Contexts

Children: Annot, AsmList, Attention, BibList, Bridge, Caution, CGraphic, D, Danger, DL, Fig, FnList, GL, L, Lines, LitData, LQ, MarkList, MkNote, MMObj, ModInfo, Note, NoteList, OL, P, ParmL, PBlk, Screen, Syntax, Table, UL, Xmp.

Parents: Proc.

ProcStep (procedure step)

Purpose

The ProcStep element describes a single step in a procedure. A procedure is made up of one or more procedure steps. Each step contains a description of the step followed by an optional decision point specification indicating what further action to take. The default action is to proceed to the next step in the procedure. A step can also contain a StepNotes element containing notes about the step.

A procedure step description can itself contain a procedure, allowing you to nest procedures to any level desired (within the general element nesting limits imposed by IBMIDDoc).

Migration Note

Bookmaster only supports three levels of nesting. For migration purposes, nesting within procedure elements should be limited to three levels.

Examples

```
<procstep><proccmnd>
<desc>Change the diaper.</desc>
</proccmnd><procexit>Johnny was uncomfortable.</procexit>
</procstep>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See Chapter 22, “Creating maintenance analysis procedures” on page 207.

Contexts

Children: DecisionPnt, ProcCmnd, ProcExit, StepNotes, TitleBlk.

Parents: Else, Proc, ProcCmnd, Then.

ProcSumm (procedure summary)

Purpose

The ProcSumm element contains procedure summary items.

Examples

```
<procsumm>
<p>And that is how you care for a child.</p>
</procsumm>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 22, “Creating maintenance analysis procedures” on page 207.

Contexts

Children: ProcSummItem.

Parents: Proc.

ProcSummItem (procedure summary item)

Purpose

The ProcSummItem element specifies a procedure summary items.

Contexts

Children: ProcEntry, ProcExit.

Parents: ProcSumm.

ProdInfo (product information)

Purpose

The ProdInfo element contains the name and version number of the product that is associated with the document.

Examples

```
<PRODINFO>
 <PRODNAME>Test Prod</PRODNAME>
 <VERSION>2</VERSION>
</PRODINFO>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Other prolog elements” on page 90.

Contexts

Children: ProdName, Version.

Parents: DProlog, Prolog, SpecDProlog.

ProdName (product name)

Purpose

The ProdName element contains the name of the product with which the document is associated.

Examples

```
<IBMPRODINFO>
 <PRODNAME>Test Prod</PRODNAME>
 <VERSION>2</VERSION>
 <REL>3</REL>
 <MOD>1</MOD>
</IBMPRODINFO>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Other prolog elements” on page 90.

Contexts

Children: text (#pcdata).

Parents: IBMLibEntry, IBMProdInfo, LibEntry, ProdInfo.

Prolog (document metainformation)

Purpose

The Prolog element contains metainformation about a document, which is information that describes the document, such as the document title, the author, and the document number. It also contains many different types of markup definitions used to define classes and properties. It contains *collectors* which contain information on reuse elsewhere in the document (using GLDefs and ObjLib).

Examples

```
<prolog>
<?xpp:lers nopage>
<ibmbibentry><doctitle>
<library><titleblk>
<title>ID Workbench</title>
</titleblk></library>
<titleblk>
<title>IBMIDDoc User's Guide and Reference</ph></title>
</titleblk></doctitle>
<ibmdocnum>SH21-0783-09</ibmdocnum>
<externalfilename>iddugref</externalfilename>
</ibmbibentry><masterindexinfo><masterindexprefix>
IDDOC</masterindexprefix></masterindexinfo>
 ...
</prolog>
```

Attributes

See “Common Element Attributes (small set)” on page 228.

Usage

See “The preface” on page 100.

Contexts

Children: Approvers, BibEntryDefs, CopyrDefs, CritDates, GlDefs, IBMBibEntry, IBMProdInfo, IdxDefs, LDescs, Maintainer, MasterIndexInfo, ObjLib, Owners, ProdInfo, PropDefs, QualifDefs, RevDefs.

Parents: IBMIDDoc.

PropDef (property set definition)

Purpose

The PropDef element defines values for properties that are common to a set of elements. Any element can refer to a PropDef element using the common PROPSRC attribute. When a set of properties applies only to a certain set of element types, the ELETYPES attribute can be used to indicate which element types can refer to a given PropDef element.

You can also use PropDef to contain PROPS definitions for property attributes that are common to all elements, such Language and Proc. See “Defining Element Properties” on page 201 for guideance on using property definitions..

Examples

```
<propdef eletypes="xmp" style="BKM:(keep=10)">
<desc>Allow examples in BookMaster output to flow,
keep the 1st 10 lines together.</desc>
</propdef>
```

Attributes

PROPNAME=*name*
Defines the name to be referenced by the PROPSRC attribute.

ELETYPES=*element names*
Defines those element types (generic identifiers) to which this PropDef applies.

Use ELETYPES when a PropDef is only meaningful for a specific set of element types, such as when the STYLE value is element-specific. When an ELETYPES value is specified, the PropDef values will apply only to elements of the specified type.

See "Common Element Attributes (small set)" on page 228.

Usage

See "Defining Element Properties" on page 201.

Contexts

Children: Desc.

Parents: PropDefs, PropGroup.

PropDefs (property definitions)

Purpose

The PropDefs element contains property definition elements. Properties apply to all elements contained within the division with which the property definitions are associated.

Examples

```
<propdefs>
<propdef eletypes="xmp" style="BKM:(keep=10)">
<desc>Allow examples in BookMaster output to flow,
keep the 1st 10 lines together.</desc>
</propdef>
 ...
</propdefs>
```

Attributes

See "Common Element Attributes (small set)" on page 228.

Usage

See Chapter 20, "Property and Class Definitions" on page 201.

Contexts

Children: ClassDef, DLDef, FigDef, GLDef, LERSDef, MkDesc, ModInfoDef, ModItemDef, MsgItemDef, MsgLDef, OLDef, PropDef, PropDesc, PropGroup, ScreenDef, SyntaxDef, ULDef, XMPDef.

Parents: DProlog, Prolog, SpecDProlog.

PropDesc (property description)

Purpose

The PropDesc element contains the definitions used in the values of the PROPS attribute. It also contains a default value that is used if no PROPS value is assigned by the author.

Examples

```
<propdesc propname="ref" default="true">
<desc>Include the Reference part</desc>
</propdesc>
```

Attributes

PROPNAME=*property-name*
: Contains the value that is being defined. This value can be used on the PROPS attribute.

Default=true | false
: Sets the default value of the property. You set the value to true or false. An unset variable is set assumed to be false.

Usage

See "Setting the properties to true or false" on page 197.

Contexts

Children: Desc, Title.

Parents: PropDefs, PropGroup.

PropGroup (property group)

Purpose

The PropGroup element enables you to organize elements within a PropDefs section. Because PropDef elements are used to define property values, the processing system cannot use those same properties to determine whether or not a given PropDef element should be processed. The PropGroup element provides a way to use properties to use or ignore PropDef elements.

The typical case is one where you want one set of properties for one output and a different set for another. You can use PropGroup elements with the PropDefs element to contain the different PropDef elements.

You can nest PropGroup elements in order to take advantage of property inheritance.

Examples

```
<PROLOG>
   ⋮
<PROPDEFS>
 <PROPDEF PROPNAME='xpert" SEC="IUO" PROPS="expert">
   <DESC>This property definition applies to all elements in the document
 </PROPDEF>
  <PROPGROUP PROPS="display">
   <DESC>The following property definition only applies
    when processing for online display.
   </DESC>
  </PROPGROUP>
  <PROPGROUP PROC="print">
   <DESC>The following property definition only applies
    when processing for print.
   </DESC>
  <PROPDEF>
```

```
   ⋮
 </PROPGROUP>
</PROPDEFS>
   ⋮
</PROLOG>
```

Attributes

See “Common Element Attributes (small set)” on page 228.

Contexts

Children: ClassDef, Desc, DLDef, FigDef, GLDef, LERSDef, MkDesc, ModInfoDef, ModItemDef, MsgItemDef, MsgLDef, OLDef, PropDef, PropDesc, PropGroup, ScreenDef, SyntaxDef, ULDef, XMPDef.

Parents: PropDefs, PropGroup.

PrtLoc (country where printed)

Purpose

The PrtLoc element contains the name of the country where a document or library was printed.

Examples

```
<PRTLOC>Boise, Idaho</PRTLOC>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “An example of using BibEntry and BibEntryDefs” on page 144.

Contexts

Children: text (#pcdata), Ph.

Parents: BibEntry, IBMBibEntry, IBMLibEntry, LibEntry.

PublicID (public identifier)

Purpose

The PublicID element contains the SGML public identifier for a document. It is intended that the public identifier of the document entity be used by presentation systems to locate the actual document, but specific presentation systems may define application-specific data to be specified as the system identifier of the document entity if they do not support the use of public identifiers. The public identifier can be included in the BibEntry itself as a way of keeping a document’s formal public identifier definition with the rest of its bibliographic information. This could allow, for example, the automatic generation of entity declarations for documents described by BibEntry elements.

See Appendix B, “Proposed IBM Standard for Formal Public Identifiers” on page 459 for a description of the formal public identifier standard defined by the IBM Corporation for its internal and external use.

Examples

```
<BIBENTRY ID="iddocref">
   ⋮
 <PUBLICID>+//ISBN 0-933186::IBM//DOCUMENT IBMIDDoc Reference//EN
   ⋮
</BIBENTRY>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Contexts

Children: text (#pcdata), Ph.

Parents: BibEntry, IBMBibEntry, IBMLibEntry, LibEntry.

Publisher (document publisher)

Purpose

The Publisher element contains the name and address of the publisher of the document.

Examples

```
<PUBLISHER>IBM CORPORATION</PUBLISHER>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “About the prolog” on page 88.

Contexts

Children: Address, CorpName.

Parents: BibEntry, IBMBibEntry, IBMLibEntry, LibEntry.

PV (parameter variable)

Purpose

The PV element contains a parameter variable. The output style specification for a PV element provides a visual cue to the user, denoting that the content of the PV element has special meaning.

Examples

```
<P>This is a description of a
<PV>parameter variable</PV>
</P>
```

PV (parameter variable)

Attributes

OPTREQ=REQ | OPT | DEF
Indicates whether or not the variable is optional. REQ (required) is the default.

See "Common Element Attributes (large set)" on page 227.

Usage

See "Highlighting" on page 43.

Contexts

Children: text (#pcdata).

Parents: AnnotBody, Attention, Bridge, Caution, CompCmt, Danger, Defn, Desc, entry, Fn, L, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, MsgText, NoteBody, P, Ph, Q, SynNote, Term, Warning, Xmp, XPh.

Q (quotation phrase)

Purpose

Use the Q element to contain material excerpted from another source and that is used within the context of a paragraph or similar element. Use LQ for quotations that contain more than one paragraph. In the default presentation style, the quoted material is presented inline with the material around it. It is usually surrounded by typographical quotes as well.

One of the purposes of Q is to identify a quotation, it can refer to a BibEntry element using the BIBID attribute.

Examples

```
⋮
<P>New presidents often try to inspire the country to face new challenges
with words like:
 <Q BIBID="jfk0161">Ask not what your country can do for you,
   ask what you can do for your country</Q>
   ⋮
```

Attributes

BibId=*bibentry_id*
The ID of the BibEntry element that defines the source of the quotation. This is optional.

"Common Element Attributes (large set)" on page 227

Usage

See "Quotes and excerpts" on page 47.

Contexts

Children: text (#pcdata), Address, APL, Bin, Char, Cit, Date, Dec, Formula, Hex, L, MD, MV, Num, Oct, Ph, PK, PV, Q, RefKey, SynPh, Term, TM, XPh, XRef.

Parents: AnnotBody, Attention, Bridge, Caution, CompCmt, Danger, Defn, Desc, entry, Fn, L, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, NoteBody, P, Ph, Q, SynNote, Warning.

Qualif (qualification)

Purpose

The Qualif element contains a qualification definition that is referenced by elements to which the qualification applies.

A qualification is a limitation or restriction on the application of information. For example, a qualification might be that information applies to 'OS/2 2.1 users only'. The QUALIF attribute is used to reference the Qualif element from the element that contains information of this type.

The Qualif element cannot be used to include or exclude information. It cannot be used for conditional processing. The QUALIF attribute and element have no effect on property-based retrieval.

For more information about the Qualif element, see "Other prolog elements" on page 90.

Examples

```
<qualifdefs>
 <qualif id="win99" ident="use">
  <title>Windows/99</title>
  <desc>Windows/99 information</desc>
 </qualif>
 <qualif id="os25" ident="use">
  <title>OS/2.5</title>
  <desc>OS/2.5 information</desc>
 </qualif>
</qualifdefs>
```

Attributes

ID=*revision_ID*
: The ID of this qualification. The ID attribute is required. The ID is referred to with the QUALIF attribute from any element.

IDENT= USE | IGNORE
: Indicates whether the qualification is a active.

"Common Element Attributes (small set)" on page 228

Usage

See "Qualifying information" on page 49.

Contexts

Children: Desc, Title.

Parents: QualifDefs.

QualifDefs (qualification definitions)

Purpose

The QualifDefs element contains a list of Qualif elements which are used in the document or division in which it is found.

Examples

```
<qualifdefs>
 <qualif id="win99" ident="use">
  <title>Windows/99</title>
  <desc>Windows/99 information</desc>
 </qualif>
 <qualif id="os25" ident="use">
  <title>OS/2.5</title>
  <desc>OS/2.5 information</desc>
 </qualif>
</qualifdefs>
```

Attributes

"Common Element Attributes (small set)" on page 228

Usage

See "Other prolog elements" on page 90.

Contexts

Children: Qualif.

Parents: DProlog, Prolog, SpecDProlog.

RCF (reader comment form)

Purpose

The RCF element contains the elements necessary to produce a reader comment form.

In order for the standard IBM RCF to be generated, the following IBMIDDoc elements must be specified in the document prolog:

- DocTitle
- Library
- IBMDocNum
- Version
- Release
- Maintainer

When creating an RCF for hardcopy, be sure to include the <MAINTAINER> section in your profile. This has the address information, fax number, etc. to be included in the RCF. Also, you need to specify the <RCF> element in the back matter.

Examples

For the RCF to be generated, you need to specify the MAINTAINTER element information in the prolog of your document:

```
<maintainer>
<corp>
<corpname>IBM Corporation</corpname>
<address>ATTN: Dept 542
3605 HWY 52 N
Rochester, MN
<postalcode>55901-9986</postalcode>
<phone equip="fax">1-800-555-1212</phone></address>
</corp>
</maintainer>
```

In the back matter, you need to include the RCF element; for example:

```
<backm>
<rcf><gendtitle></rcf>
</backm>
```

Attributes

"Common Element Attributes (small set)" on page 228

Usage

See "Using reader's comment form (RCF)" on page 103.

Contexts

Children: GendTitle, RetKey, TitleBlk.

Parents: BackM, FrontM.

RefKey (reference key)

Purpose

The RefKey element establishes a visual link that identifies a specific part of a graphic.

Examples

```
See <refkey>1</refkey> for that part.
```

Attributes

"Common Element Attributes (large set)" on page 227

Usage

See "Highlighting" on page 43.

Contexts

Children: text (#pcdata).

Parents: AnnotBody, Attention, Bridge, Caution, CGraphic, CompCmt, Danger, Defn, Desc, entry, Fig, FigSeg, Fn, L, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, NoteBody, P, Ph, Q, Screen, SynNote, Term, Warning, Xmp.

Release (product release identifier)

Purpose

The Release element contains the product release number.

Examples

```
<VERSION>1</VERSION>
<RELEASE>1</RELEASE>
<MOD>1</MOD>
```

Attributes

"Common Element Attributes (large set)" on page 227

Usage

See "Using IBMProdInfo" on page 92.

Contexts

Children: text (#pcdata), Ph.

Parents: IBMProdInfo.

RepSep (syntax repeat separator)

Purpose

The RepSep element defines a repeat separator within a syntax definition. Repeat separators must be defined at the beginning of the syntax definition.

Examples

```
<syntax>
<repsep id="rsep0003a"></repsep>
<group repid="rsep0003a">
<var>variable</var>
</group>
</syntax>
```

Attributes

OPTREQ=OPT | REQ
Indicates whether the contained group is required or optional.

CONVAR=CONSTANT|VAR
Indicates whether the content of the element is a constant or a variable in the context where is it used.

See "Common Element Attributes (large set)" on page 227.

Usage

See Chapter 15, "Programming Syntax Diagrams" on page 147.

Contexts

Children: text (#pcdata).

Parents: SynBlk, Syntax.

RetKey (retrieval key)

Purpose

RetKey can be used to specify subject heading retrieval aid text and graphics for those output types which support such graphics. This can also contains information to be used by an information management system. Using this key, the system could conduct queries without resorting to a full-text search of all of the information. Use the RetKey element to contain a list of meaningful terms and abbreviations that might be used as keywords during a database search.

Examples

```
<D>
<DPROLOG>
<TITLEBLK><TITLE>Configuring Your New Whantoozler
</TITLE></TITLEBLK>
<RETKEY>Whantoozler 5.0 Setup</RETKEY>
</DPROLOG>
 ...
```

Attributes

OBJ=*graphic_entity*
specifies the the name of the graphic to be used as a retrieval aid.

See "Common Element Attributes (large set)" on page 227.

Contexts

Children: text (#pcdata), Ph.

Parents: Appendix, DProlog, FigList, IBMBibEntry, IBMSafety, Index, LE, LERS, Mod, ModInfo, MsgList, PartAsm, PNIndex, Proc, RCF, Safety, SpecDProlog, TList, TOC.

Rev (revision)

Purpose

The Rev element defines a single revision for a document or division. The REV attribute is used on revised elements to refer to the applicable Rev element. Use the Desc element within Rev to describe the purpose of the revision.

Examples

```
<prolog>
 ...
<revdefs>
<rev id="v4r5" ident="use">
<date>9/9/99</date>
<desc>First draft for v4r5</desc>
```

```
</rev>
</revdefs>
 ...
</prolog>
```

Attributes

ID=*revision_ID*
: The ID of this revision. The ID attribute is required. The ID is referred to with the REV attribute from any element.

IDENT= USE | IGNORE
: Indicates whether the revision is an active revision. Revisions that are active (USE) will be indicated by whatever mechanism is defined in the presentation style, typically by placing a vertical bar or other character in the margin.

  Revisions for which IGNORE is specified are ignored when the document is processed.

CODE=*character*
: Associates a character with the revision. When no code is specified, you get a vertical bar to the left of text containing the active REV attribute. This is what you want to appear in the final edition. For internal drafts, you can use some other character to indicate the revision level. For example, a plus sign, an asterisk, or a number. Use only a single character.

REASON=*reason_text*
: Contains the text explaining the reason for the revision.

See "Common Element Attributes (small set)" on page 228.

Usage

See "Using Revisions" on page 109.

Contexts

Children: Author, Date, Desc.

Parents: RevDefs.

RevDefs (revision tracking information)

Purpose

The RevDefs element contains Rev and Mark elements which define the revision history of the document or division.

Examples

```
<prolog>
 ...
<revdefs>
<rev id="v4r5" ident="use">
<date>9/9/99</date>
<desc>First draft for v4r5</desc>
</rev>
</revdefs>
 ...
</prolog>
```

Attributes

See “Common Element Attributes (small set)” on page 228.

Usage

See “Using Revisions” on page 109.

Contexts

Children: Mark, Rev.

Parents: DProlog, Prolog, SpecDProlog.

Row (table row)

Purpose

The Row element contains the row information for a row in a TGroup.

Examples

```
<table pgwide="0" id="tablesample">
<cap>Sample table caption</cap>
<tgroup cols="1">
<colspec colname="col1">
<tbody>
<row>
<entry colname="col1">my little</entry>
</row>
<row>
<entry colname="col1">sample table</entry>
</row>
</tbody>
</tgroup>
</table>
```

Attributes

ROWSEP=0 | 1
This attribute’s value specifies that a row separator rule should be:
- displayed below each Entry element ending in a Row (1)
- not displayed at all (0)

Shade=NOShade | XLight | Light | Meduim | Dark | XDark
Use the Shade attribute to specify the shading. This table example shows the shading values used in table cells:

Table 22. Cell entry shading. In the editor, use the modify attributes icon or Ctrl-A to set the Shade attribute for the cell’s Entry tag.

noshade (0%)	xlight (5%)	light (26%)	medium (50%)	dark (74%)	xdark (100%)
the	quick brown	fox	jumps over	the lazy	■

VALIGN=TOP | MIDDLE | BOTTOM
This attribute specifies the vertical alignment of the text contained in the Entry elements.

TOP
specifies alignment of the text at the top of the Entry elements.

MIDDLE
specifies alignment of the text at the middle of the Entry elements.

BOTTOM
specifies alignment of the text at the bottom of the Entry elements (the default).

See "Common Element Attributes (large set)" on page 227.

Usage

See Chapter 7, "Creating IBMIDDoc Tables" on page 67.

Contexts

Children: entry.

Parents: tbody, tfoot, thead.

Safety (safety notices)

Purpose

Use Safety to contain or refer to any safety-related information such as cautions, warnings, and FCC notices.

Examples

```
<safety><titleblk><title>Safety Notices</title></titleblk>
<caution>Watch out for splinters.</caution>
</safety>
```

Attributes

| **LAYOUT=Document-Layout | OneCol | TwoCol | OffsetCol**
| Specifies the column-style for this portion of the book.

| **Document-Layout**
| The section uses the default layout for the document style.

| **OffsetCol**
| Formats the text using an indented, one-column layout. The headings
| format across the page, with the offset text, or indented from the
| margin.

| **OneCol**
| The text formats across the entire page.

| **TwoCol**
| The text formats in two columns.

SPEC=AUTO | MAN
Specifies whether the content of the element is generated. If SPEC=AUTO is specified, the element's content is generated. This attribute is not supported at this time.

See "Common Element Attributes (large set)" on page 227.

Contexts

Children: Attention, Caution, Danger, DL, Fig, GendTitle, L, MMObj, Note, NoteList, OL, P, PBlk, RetKey, Table, TitleBlk, UL, Warning.

Parents: FrontM.

Screen (display screen)

Purpose

The Screen element contains or refers to a representation of a computer screen or user interface panel (window).

Examples

```
<screen>
cpyf      CopyFile Command

From file . . . . ___________

To file . . . . . ___________

F1=Help  3=Exit  12=Cancel
</screen>
```

Attributes

OBJ=*data_entity_name*
: Refers to an SGML data entity that contains the screen representation. The data entity can be in any supported notation and may in fact be a reference to a live version of the panel if that is supported by your online presentation system.

  When the OBJ attribute is specified, it is an error to specify any screen data or the Screen end tag.

NOTATION=
: Defines the notation of the contained data for inline screen data, as follows:

  LINESPEC
  : Significant record ends are preserved in the output.

| **LINELENGTH=***characters*
: | This specifies the number of characters in widest line. If the number of characters does not fit in the column or page width, the text size is reduced. If possible, when LINELENGTH is specified, the width of the screen border is shrunk to fit around the specified number of characters.

  | If no LINELENGTH is specified, the formatter uses the default size for screens for that style (or the current document point size if it is smaller) and makes the screen fit the indented area (or the page area if pgwide=1).

  | If LINELENGTH is specified, the formatter starts scaling down from the current document point size. If number of characters fits, it makes the screen just large enough to fit them and shrinks the screen to fit around them. If number of characters doesn’t fit, it scales down the characters to fit and has the screen fit around them.

| **PGWIDE=1 | 2**
: | This specifies the width of the screen. 1 is for a page-wide screen; 2 indents to the current indention in the column (2 is the default)

DEF=*definition-name*
: Specifies the definition to be used. This points to a DefName attribute on a corresponding DEF tag. The settings defined on that DEF tag are used as defaults for this tag. See Chapter 9, “Using definition tags” on page 105 for more information.

See “Common Element Attributes (large set)” on page 227.

Usage

See “Screens” on page 59.

Contexts

Children: text (#pcdata), L, LitData, MMObj, Ph, RefKey, Term.

Parents: AnnotBody, Attention, Bridge, Caution, Danger, DBody, Defn, DIntro, DSum, entry, Fig, FigSeg, Fn, LEDI, LI, LQ, MkNote, ModDesc, ModItem, MsgItem, NoteBody, P, PBlk, ProcIntro, SynNote, Warning.

ScreenDef (Screen definition)

Purpose

The ScreenDef element sets attribute defaults for screens. ScreenDef goes within the document prolog to set definitions for the entire document; or within a division prolog to set definitions for just that division. The ScreenDef tag goes inside a PropDefs tag.

Examples

```
<screendef defname="widescreen" linelength="127" pgwide="1">
```

Attributes

DEFNAME=*definition-name*
: The DefName attribute identifies a definition element. DefName values must be unique within a single document. A element that has a DEF attribute can point a corresponding tag with a DefName attribute. DefName values can be up to 64 characters long. DefName values must start with an alphabetic character and can contain letters, numbers, dashes (-), or periods (.).

  Definition tags can be specified without a DefName attribute in the document’s prolog. This allows you to change the initial settings of attributes for the entire document when the DEF tag is in the document’s prolog. In a division’s prolog, it changes the initial settings for that division.

LINELENGTH=*characters*
: This specifies the number of characters in widest line. If the number of characters does not fit in the column or page width, the text size is reduced. If possible, when LINELENGTH is specified, the width of the screen border is shrunk to fit around the specified number of characters.

  If no LINELENGTH is specified, the formatter uses the default size for screens for that style (or the current document point size if it is smaller) and makes the screen fit the indented area (or the page area if pgwide=1).

  If LINELENGTH is specified, the formatter starts scaling down from the current document point size. If number of characters fits, it makes the screen just large enough to fit them and shrinks the screen to fit around them. If number of characters doesn’t fit, it scales down the characters to fit and has the screen fit around them.

PGWIDE=1 | 2
This specifies the width of the screen. 1 is for a page-wide screen; 2 indents to the current indention in the column (2 is the default)

Props=*properties*
The Props attribute is used to specify the condition under which the information contained within the element appears. See “Property-Based Retrieval” on page 195.

Usage

See Chapter 9, “Using definition tags” on page 105.

Contexts

Children: empty.

Parents: PropDefs, PropGroup.

Sem (semantic meaning)

Purpose

The Sem element defines the meaning of a class. It should describe the type of information for which the class should be used.

Attributes

See “Common Element Attributes (large set)” on page 227.

Contexts

Children: text (#pcdata), DL, Fig, L, MMObj, Note, NoteList, OL, P, PBlk, Ph, Table, Term, TM, UL.

Parents: ClassDef.

Sep (syntactic separator)

Purpose

The Sep element contains a separator that is to separate keywords, variables, operators, or groups.

Examples

```
<syntax>
<group>
<kwd>FRED</kwd>
<sep>,</sep>
<kwd>BARNEY</kwd>
</group>
</syntax>
```

Attributes

OPTREQ = REQ | OPT
Indicates whether or not the separator is optional or required. REQ (required) is the default. Any separator that is not optional is, by definition, required.

CONVAR=CONSTANT | VAR
Indicates whether the content of the element is a constant or a variable in the context where is it used.

LINKEND=*id*
Contains an ID reference that enables a link to an arbitrary location in the document.

See "Common Element Attributes (large set)" on page 227.

Usage

See Chapter 15, "Programming Syntax Diagrams" on page 147.

Contexts

Children: text (#pcdata).

Parents: Group, SynPh.

SOA (summary of amendments)

Purpose

The SOA element contains information summarizing any changes made to the information since prior versions.

Examples

```
<soa>
 <specdprolog><titleblk><title>What's new and different
  </title></titleblk></specdprolog>
 <dbody>
  <p>Changes since the last edition include...</p>
 </dbody>
</soa>
```

Attributes

| **LAYOUT=Document-Layout | OneCol | TwoCol | OffsetCol**
| Specifies the column-style for this portion of the book.

| **Document-Layout**
| The section uses the default layout for the document style.

| **OffsetCol**
| Formats the text using an indented, one-column layout. The headings
| format across the page, with the offset text, or indented from the
| margin.

| **OneCol**
| The text formats across the entire page.

| **TwoCol**
| The text formats in two columns.

Toc=toc | notoc
Specifies whether the heading should appear in the table of contents. The default is for headings to appear in the table of contents; to a certain level (such as heading level 3). This replaces the older style=hidden attribute. Use this to prevent headings from appearing in the table of contents. It cannot be used to add a heading to the table of contents that would not normally appear. The levels of headings that appear in the table of contents are determined by the MaxToc attribute of the IBMIDDoc tag.

See "Common Element Attributes (large set)" on page 227.

Usage

See "Summary of changes" on page 100.

Contexts

Children: DBody, DIntro, DSum, SpecDProlog.

Parents: BackM, FrontM.

SpanSpec (span specification)

Purpose

In tables, this specifies how cells are to be combined.

Examples

```
<table frame="all" pgwide="0" id="complxt">
<cap>Complex table example</cap>
<tgroup cols="3" colsep="1" rowsep="1">
<colspec colname="col1" colwidth="25*">
<colspec colname="col2" colwidth="32*">
<colspec colname="col3" colwidth="38*">
<spanspec namest="col1" nameend="col2" spanname="1to2">
<tbody>
<row>
<entry spanname="1to2" valign="top">Row 1, Cell 1
</entry>
<entry colname="col3" morerows="1" valign="top">Row
1, Cell 2</entry>
</row>
<row>
<entry valign="top">Row 1, Cell 3</entry>
<entry valign="top">Row 1, Cell 4</entry>
</row>
</tbody>
</tgroup>
</table>
```

Attributes

namest=*starting column*

nameend=*ending column*

spanname=*name*

align= center | char | justify | left | right
How to align the content of the cell.

charoff=*distance*
Offset from left edge of cell.

char
Character to align to

colsep= 1 | 0
Draw a line to the right of the cell (1); or not (0).

rowsep = 1 | 0
Draw a line under the bottom of the cell (1); or not (0).

Usage

See Chapter 7, "Creating IBMIDDoc Tables" on page 67.

Contexts

Children: empty.

Parents: tgroup.

SpecDProlog (special section division prolog)

Purpose

The SpecDProlog element contains prolog information specific to a division.

Examples

```
<SPECDPROLOG>
 <GENDTITLE>
  <AUTHORS>
   <AUTHOR>
    <PERSON>
     <NAME>
      Rick Dennis
     </NAME>
    </PERSON>
   </AUTHOR>
  </AUTHORS>
</SPECDPROLOG>
```

Attributes

See "Common Element Attributes (small set)" on page 228.

Contexts

Children: Approvers, Authors, BibEntryDefs, CopyrDefs, CritDates, GendTitle, GlDefs, IBMProdInfo, IdxDefs, LDescs, Maintainer, MasterIndexInfo, MetaData, ObjLib, Owners, ProdInfo, PropDefs, QualifDefs, RetKey, RevDefs, TitleBlk.

Parents: Abbrev, Abstract, Bibliog, Glossary, Legend, MasterIndex, Preface, SOA.

StepNotes (step notes)

Purpose

Use StepNotes to contain notes about a procedure step, such as special considerations about the step. Do not use StepNotes to convey decision-making information. Rather, use DecisionPnt to identify conditions and actions that may change the path through the procedure.

Examples

```
<procstep>
<stepnotes>
<li>Johnny can eat solid food.</li>
<li>Continue at <xref refid="frozstk"></li>
</stepnotes></procstep>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 22, “Creating maintenance analysis procedures” on page 207.

Contexts

Children: Bridge, LI, LIBlk.

Parents: ProcStep.

StepRef (procedure step reference)

Purpose

The StepRef element cross-references to a step in a procedure (PROC).

Contexts

Children: empty.

Parents: Desc, ProcEntry, ProcExit.

STitle (shortened title)

Purpose

The STitle element contains a shortened title for an element or document. The STitle is used in running footers by some output formatter styles.

For XHTML and HTML, you can use the STitle to override the text used in the table of contents for use in Informaiton Centers. This also requires the use of the /TOCSTITLE formatting option.

Examples

```
<ibmbibentry>
<doctitle><titleblk>
<title>Document Title</title>
<stitle>Short title, used for running foot</stitle>
<subtitle>Subtitle, using on title page</subtitle>
</titleblk></doctitle></ibmbibentry>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Contexts

Children: text (#pcdata), L, Ph, Term, TM.

Parents: TitleBlk.

SubTitle (descriptive subtitle)

Purpose

The SubTitle element contains a descriptive subtitle that can be used on title pages to further describe the document's subject matter.

SubTitle is also allowed with in heading and division titles, but it should not be used.

Examples

```
<ibmbibentry>
<doctitle><titleblk>
<title>Document Title</title>
<stitle>Short title, used for running foot</stitle>
<subtitle>Subtitle, using on title page</subtitle>
</titleblk></doctitle></ibmbibentry>
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Contexts

Children: text (#pcdata), L, Ph, Term, TM.

Parents: TitleBlk.

SynBlk (syntax block)

Purpose

The SynBlk element organizes syntax definitions into titled subdivisions. Use syntax blocks to organize the elements of a syntax definition into logical, optionally titled groupings. For example, a single command may have several forms. Syntax blocks allow you to define all the forms in a single syntax definition.

In hardcopy, syntax blocks also automatically scale the diagram portion they contain to fit the column width.

Examples

```
<syntax>
<synblk>
<group>
<kwd>FORM</kwd>
<kwd>PROC</kwd>
</group>
<group>
<kwd>FILE</kwd>
<kwd>NAME</kwd>
</group>
</synblk>
</syntax>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 15, “Programming Syntax Diagrams” on page 147.

Contexts

Children: Fragment, FragRef, Group, RepSep, SynNote, Title.

Parents: Syntax.

SynNote (syntax note)

Purpose

The SynNote element contains a note within a syntax definition group or fragment. Use SynNote to add notes to your syntax definition to explain aspects of the syntax that cannot be expressed in the syntax markup itself. In the default presentation, the syntax notes are associated with the syntax diagram using numeric callouts and the syntax notes are themselves collected at the end of the presented syntax diagram or syntax block.

Examples

```
<syntax>
<group>
<kwd>FRED</kwd>
<synnote>This is a rather common name.</synnote>
</group>
</syntax>
```

Attributes

refid=*identifier*
: Refers the the ID of a corresponding SYNNOTE tag. This allows you to enter the note text once in the diagram, and reuse the note for another item in the diagram.

callout=*character*
: This allows you to specify one character to use instead of a number for indicating the note.

See “Common Element Attributes (large set)” on page 227.

SynNote (syntax note)

Usage

See “Syntax Notes” on page 156.

Contexts

Children: text (#pcdata), Address, Annot, APL, Bin, Bridge, CGraphic, Char, Cit, Date, Dec, DL, Fig, Formula, GL, Hex, L, Lines, LitData, LQ, MD, MMObj, ModInfo, MV, Note, NoteList, Num, Oct, OL, P, ParmL, PBlk, Ph, PK, PV, Q, RefKey, Screen, SynPh, Table, Term, TM, UL, Xmp, XPh, XRef.

Parents: Fragment, Group, SynBlk, Syntax.

SynPh (syntax phrase)

Purpose

The SynPh element contains syntax definition elements and is used in the context of the information around it. Use SynPh to present syntax fragments outside the context of a complete syntax definition.

Examples

```
<synph><kwd optreq="def">Filename</kwd></synph>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Syntax Phrases” on page 157.

Contexts

Children: text (#pcdata), Delim, Kwd, Oper, Sep, Var.

Parents: AnnotBody, Attention, Bridge, Caution, CompCmt, Danger, Defn, Desc, entry, Fn, L, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, MsgText, NoteBody, P, Ph, Q, SynNote, Term, Warning, Xmp.

Syntax (syntax diagram)

Purpose

The Syntax element contains the definition of the syntax of a statement in some computer language; or a command, function call, programming language statement, or other such construct. Use syntax definitions to define the rules for creating statements in some computer language, for example commands in a command language or programming language expressions. Syntax definitions can also be used to model more abstract constructs, such as the structural relationships between elements in a language.

The default presentation style is as “railroad tracks”, but their presentation is not limited to that form. A given definition can be presented in a variety of ways.

Examples

```
<syntax><title>Database Reference</title>
<repsep id="rsep0006"></repsep>
<group>
<kwd>CREATE TABLE</kwd>
</group>
<group>
<var>table_name</var>
</group>
<group repid="rsep0006">
<group style="bkm:(composite)">
<delim startend="START">(</delim>
<var>column_name</var>
</group>
<fragref><title>Data Type</title></fragref>
<kwd optreq="OPT">NOT NULL</kwd>
<delim optreq="req" startend="END">)</delim>
</group>
<fragment><title>Data Type</title>
<group choiceseq="CHOICE">
<kwd>INTEGER</kwd>
<group>
<group choiceseq="CHOICE">
<kwd>DECIMAL</kwd>
<kwd>DEC</kwd>
</group>
<group style="BKM:(COMPOSITE)">
<delim startend="START">(</delim>
<var>length</var>
<sep>&ssbl;+&ssbl;</sep>
<var>colwidth</var>
<delim startend="END">)</delim>
</group>
</group>
<group>
<group choiceseq="CHOICE">
<kwd>CHARACTER</kwd>
<kwd>CHAR</kwd>
</group>
<group optreq="OPT" style="BKM:(COMPOSITE)">
<delim startend="START">(</delim>
<var>length</var>
<delim startend="END">)</delim>
</group>
</group>
<group style="BKM:(COMPOSITE)">
<kwd>GRAPHIC</kwd>
<delim startend="START">(</delim>
<var>length</var>
<delim startend="END">)</delim>
</group>
</group>
</fragment>
</syntax>
```

Attributes

SYNSTYLE=SPACE | LBLBOX | BOX | RULE
Causes the figure to have a frame. The default is space — no frame.

LblBox
Causes a box to be placed around the diagram. The top line of the box has text label that is taken from the diagram's Title tag.

Box Causes a box to be placed around the diagram.

Rule Causes a line to be placed above and below the diagram; to visually separate it from the surrounding text.

PGWIDE=1 | 2
This specifies the width of the syntax diagram. 1 creates a page-wide diagram; 2 indents to the current indention in the column (2 is the default).

ScalePct=*scale-percent*
You can use the ScalePct attribute to scale the text up or down in the element. The scale-percent is a positive, whole number. 100 is the normal size; 50 is 50% smaller; 200 is 200% larger. For example, this scales the text to 80% of the body text:

```
scalepct="80"
```

DEF=*definition-name*
Specifies the definition to be used. This points to a DefName attribute on a corresponding DEF tag. The settings defined on that DEF tag are used as defaults for this tag. See Chapter 9, "Using definition tags" on page 105 for more information.

Complang
Specifies the computer language. This is optional.

See "Common Element Attributes (large set)" on page 227.

Usage

See Chapter 15, "Programming Syntax Diagrams" on page 147.

Contexts

Children: Fragment, FragRef, Group, RepSep, SynBlk, SynNote, Title.

Parents: AnnotBody, Attention, Bridge, Caution, Danger, DBody, Defn, DIntro, DSum, entry, Fig, FigSeg, Fn, LEDI, LI, LQ, MkNote, ModDesc, ModItem, MsgItem, NoteBody, P, PBlk, ProcIntro, Warning.

SyntaxDef (Syntax definition)

Purpose

The SyntaxDef element sets attribute defaults for syntax diagrams. SyntaxDef goes within the document prolog to set definitions for the entire document; or within a division prolog to set definitions for just that division. The SyntaxDef tag goes inside a PropDefs tag.

Examples

```
<syntaxdef defname="widediag" synstyle="box" pgwide="1" scalepct="90">
```

Attributes

DEFNAME=*definition-name*
The DefName attribute identifies a definition element. DefName values must be unique within a single document. A element that has a DEF attribute can point a corresponding tag with a DefName attribute. DefName values can be up to 64 characters long. DefName values must start with an alphabetic character and can contain letters, numbers, dashes (-), or periods (.).

Definition tags can be specified without a DefName attribute in the document's prolog. This allows you to change the initial settings of attributes for the entire document when the DEF tag is in the document's prolog. In a division's prolog, it changes the initial settings for that division.

SYNSTYLE=SPACE | LBLBOX | BOX | RULE
Causes the figure to have a frame. The default is space — no frame.

LblBox
Causes a box to be placed around the diagram. The top line of the box has text label that is taken from the diagram's Title tag.

Box Causes a box to be placed around the diagram.

Rule Causes a line to be placed above and below the diagram; to visually separate it from the surrounding text.

ScalePct=*scale-percent*
You can use the ScalePct attribute to scale the text up or down in the element. The scale-percent is a positive, whole number. 100 is the normal size; 50 is 50% smaller; 200 is 200% larger. For example, this scales the text to 80% of the body text:

```
scalepct="80"
```

PGWIDE=1 | 2
This specifies the width of the syntax diagram. 1 creates a page-wide diagram; 2 indents to the current indention in the column (2 is the default).

Props=*properties*
The Props attribute is used to specify the condition under which the information contained within the element appears. See "Property-Based Retrieval" on page 195.

Usage

See Chapter 9, "Using definition tags" on page 105.

Contexts

Children: empty.

Parents: PropDefs, PropGroup.

Table

Purpose

The Table element contains elements that make up a IBMIDDoc table.

Examples

```
<table frame="all" pgwide="0" id="complxt">
<cap>Complex table example</cap>
<tgroup cols="3" colsep="1" rowsep="1">
<colspec colname="col1" colwidth="25*">
<colspec colname="col2" colwidth="32*">
<colspec colname="col3" colwidth="38*">
<spanspec namest="col1" nameend="col2" spanname="1to2">
<tbody>
<row>
<entry spanname="1to2" valign="top">Row 1, Cell 1
</entry>
```

```
<entry colname="col3" morerows="1" valign="top">Row
1, Cell 2</entry>
</row>
<row>
<entry valign="top">Row 1, Cell 3</entry>
<entry valign="top">Row 1, Cell 4</entry>
</row>
</tbody>
</tgroup>
</table>
```

Attributes

TOCENTRY= 0(NO) | 1(YES)
If YES, and if the Title element is included, the table title will be included in the generated TList for the document.

FRAME=TOP | BOTTOM | TOPBOT | ALL | SIDES | NONE
This attribute's value describes the frame around the table.

COLSEP=0(NO) | 1(YES)
This attribute's value specifies that the internal column rules should be:

- drawn to the right of each cell's content (1)
- not displayed at all (0)

ROWSEP 0(NO) | 1(YES)
This attribute's value specifies that the internal row rules should be:

- drawn below each Entry element that ends a row (1)
- not displayed at all (0)

ORIENT=PORT | LAND
This attribute specifies whether the orientation of the table presentation is portrait or landscape. The default is PORT for portrait.

PGWIDE= 0 | 1 | 2
This attribute's value specifies that the table width should be:

- the full page width (1)
- column width (0)
- The width of the current textline (2). Use this to have a table inside a list item format to the indentation of that list item.

| **RowHeader=FirstCol | NoRowHeader**
| This specifies whether the first column is a row header. If your table's first
| column is really a row-header, specify the RowHeader=FirstCol setting. In the
| same way that a column header introduces a table column; the row header
| introduces the table row. This is to help make tables, whose first column is a
| row-header, to be more accessible when the output is for XHTML. The default
| is NoRowHeader. Here's an example of a table where the FirstCol attribute
| should be used:

Switch Location	Setting
Hallway	On
Kitchen	Off
Bedroom	On

| And the markup:

```
<table pgwide="2" rowheader="firstcol">
<tgroup cols="2">
<colspec colname="col1">
```

```
<colspec colname="col2">
<thead>
<row>
<entry colname="col1" valign="top">Switch Setting
</entry>
<entry colname="col2" valign="top">Value</entry>
</row>
</thead>
<tbody>
<row>
<entry colname="col1">Hall switch</entry>
<entry colname="col2">On</entry>
</row>
<row>
<entry colname="col1">Kitchen switch</entry>
<entry colname="col2">Off</entry>
</row>
<row>
<entry colname="col1">Bedroom switch</entry>
<entry colname="col2">On</entry>
</row>
</tbody>
</tgroup>
</table>
```

Shade=NOShade | XLight | Light | Meduim | Dark | XDark
Use the Shade attribute to specify the shading. This table example shows the shading values used in table cells:

Table 23. Cell entry shading. In the editor, use the modify attributes icon or Ctrl-A to set the Shade attribute for the cell's Entry tag.

noshade (0%)	xlight (5%)	light (26%)	medium (50%)	dark (74%)	xdark (100%)
the	quick brown	fox	jumps over	the lazy	■

ScalePct=*scale-percent*
You can use the ScalePct attribute to scale the text up or down in the element. The scale-percent is a positive, whole number. 100 is the normal size; 50 is 50% smaller; 200 is 200% larger. For example, this scales the text to 80% of the body text:

```
scalepct="80"
```

See "Common Element Attributes (large set)" on page 227.

Usage

See Chapter 7, "Creating IBMIDDoc Tables" on page 67.

Contexts

Children: Cap, Desc, tgroup.

Parents: AnnotBody, Attention, BackCover, Bridge, Caution, Cond, Copyr, Danger, DBody, Defn, DIntro, DSum, EdNotices, Fn, FrontCover, LEDesc, LEDI, LI, LQ, MsgItem, NItem, NoteBody, Notices, P, PBlk, ProcEntry, ProcExit, ProcIntro, Safety, Sem, SynNote, TextAlt, Warning.

TBody (table body)

Purpose

The TBody element contains the body of a TGroup; the main part of a table..

Examples

```
<table frame="all" pgwide="0" id="complxt">
<cap>Complex table example</cap>
<tgroup cols="3" colsep="1" rowsep="1">
<colspec colname="col1" colwidth="25*">
<colspec colname="col2" colwidth="32*">
<colspec colname="col3" colwidth="38*">
<spanspec namest="col1" nameend="col2" spanname="1to2">
<tbody>
<row>
<entry spanname="1to2" valign="top">Row 1, Cell 1
</entry>
<entry colname="col3" morerows="1" valign="top">Row
1, Cell 2</entry>
</row>
<row>
<entry valign="top">Row 1, Cell 3</entry>
<entry valign="top">Row 1, Cell 4</entry>
</row>
</tbody>
</tgroup>
</table>
```

Attributes

VALIGN=TOP | MIDDLE | BOTTOM
This attribute specifies the vertical alignment of the text contained in the Entry elements.

TOP
specifies alignment of the text at the top of the Entry elements.

MIDDLE
specifies alignment of the text at the middle of the Entry elements.

BOTTOM
specifies alignment of the text at the bottom of the Entry elements (the default).

See “Common Element Attributes (large set)” on page 227.

Contexts

Children: row.

Parents: tgroup.

Term

Purpose

The Term element contains a term, usually within a glossary entry. When used outside the context of GLEntry, Term identifies a term that has been defined elsewhere.

Examples

```
<GLENTRY>
 <TERM>apple</TERM>
 <DEFN>The fruit of the apple tree.</DEFN>
</GLENTRY>
   ⋮
An <term>apple</term> a day keeps the doctor away.
```

Attributes

TERMDEF=*defnid*
: Contains the ID of the correct definition for the term contained in the Term element. This definition is found in the glossary or definition list markup.

See “Common Element Attributes (large set)” on page 227.

Contexts

Children: text (#pcdata), APL, Bin, Char, Dec, Formula, Hex, L, MMObj, MV, Num, Oct, Ph, PK, PV, RefKey, SynPh, Term, TM, XPh.

Parents: Address, AnnotBody, Attention, Bridge, Cap, Caution, CGraphic, CI, CLE, CompCmt, Cond, Copyr, Danger, Defn, DefnHd, Desc, DLEntry, entry, Fn, GLEntry, L, LEDesc, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, ModName, MsgText, NoteBody, P, Parm, Ph, ProcEntry, ProcExit, Q, Screen, Sem, STitle, SubTitle, SynNote, Term, TermHd, TextAlt, Title, TM, Warning, Xmp, XPh.

TermHd (term heading)

Purpose

The TermHd element contains the heading for the term portion of a definition or parameter list.

Examples

```
<DL>
 <TERMHD>Term</TERMHD>
 <DEFNHD>Definition</DEFNHD>
  <DLENTRY>
   <TERM>Red Otter</TERM>
   <DEFN>The Red Otter lives in....
   </DEFN>
  </DLENTRY>
  <DLENTRY>
   <TERM>Blue Otter</TERM>
   <DEFN>Blue Otters inhabit the....
   </DEFN>
  </DLENTRY>
</DL>
```

```
<PARML>
 <TERMHD>Parameter</TERMHD>
 <DEFNHD>Purpose</DEFNHD>
  <PARM>
   <TERM>D</TERM>
   <DEFN>The D element contains a hierarchical division.</DEFN>
  </PARM>
  <PARM>
```

```
   <TERM>P</TERM>
   <DEFN>Contains a paragraph</DEFN>
  </PARM>
</PARML>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Definition lists” on page 32.

Contexts

Children: text (#pcdata), L, Ph, Term, TM.

Parents: DL, ParmL.

TextAlt (text alternative)

Purpose

The TextAlt element contains a text description of a multimedia object for use in non-graphic environments.

Examples

```
<MMOBJ><OBJREF OBJ="TESTGRAF">
 <TEXTALT><P>This is a description of the object referred to.
 </TEXTALT>
</MMOBJ>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Including artwork in documents” on page 55.

Contexts

Children: text (#pcdata), DL, L, Note, OL, P, PBlk, Ph, Table, Term, UL.

Parents: AreaDef, MMObj.

TFoot (table footer)

Purpose

The TFoot element contains the footer rows that occur after the TBody element. TFoot cannot be created within the current version of the graphical table editor. You can add the TFoot element to your table from the tag view, but the table cannot be edited with the graphical table editor without removing the TFoot element. For this reason, using TFoot is not recommended. Use the last row of the table body to contain the table footing; typically to contain a list of table notes.

If you want to use the TFoot element, do not add it to your table markup until the rest of the table markup and content is complete.

Attributes

VALIGN=TOP | MIDDLE | BOTTOM
This attribute specifies the vertical alignment of the text contained in the Entry element.

TOP
specifies alignment of the text at the top of the Entry.

MIDDLE
specifies alignment of the text at the middle of the Entry.

BOTTOM
specifies alignment of the text at the bottom of the Entry (the default).

Usage

See "Adding footnotes to a table" on page 80.

Contexts

Children: row.

Parents: tgroup.

TGroup (table group)

Purpose

The TGroup element contains elements that make up a section of a table.

Examples

```
<TABLE FRAME="ALL">
 <TGROUP COLS="5" COLSEP="1" ROWSEP="1" ORIENT="PORT" PGWIDE="0">

    ⋮
```

Attributes

COLS=*number_of_cols*
This value indicates the number of columns defined for the TGroup.

COLSEP=0(NO) | 1(YES)
This attribute's value specifies that the internal column rules should be:
- drawn to the right of each cell's content (1)
- not displayed at all (0)

ROWSEP 0(NO) | 1(YES)
This attribute's value specifies that the internal row rules should be:
- drawn below each Entry element that ends a row (1)
- not displayed at all (0)

ALIGN=LEFT | RIGHT | CENTER | JUSTIFY | CHAR
This attribute specifies the horizontal positioning of the text:

LEFT
specifies left alignment (the default)

RIGHT
specifies right alignment

CENTER
specifies center alignment

JUSTIFY
specifies justification of the text

CHAR
specifies alignment on a particular character

PGWIDE= 0 | 1 | 2
This attribute's value specifies that the TGroup width should be:
- the full page width (1)
- column width (0)
- text-line width (2)

ColSpec
Contains the column specification for a column.

SpanSpec
Contains the specification for a table span.

THead
Contains the table header.

TFoot
Contains the table footer.

Tbody
Contains the body of a TGroup in a Table.

Usage

See Chapter 7, “Creating IBMIDDoc Tables” on page 67.

Contexts

Children: colspec, spanspec, tbody, tfoot, thead.

Parents: Table.

THead (table heading)

Purpose

The THead element contains the heading rows of a TGroup element.

Examples

```
<table frame="all" pgwide="0">
<cap>Another sample table</cap>
<tgroup cols="4" colsep="1" rowsep="1">
<colspec colname="col1" colwidth="1*">
<colspec colname="col2" colwidth="2*">
<colspec colname="col3" colwidth="3*">
<colspec colname="col4" colwidth="1*">
<thead>
<row>
<entry valign="top" rowsep="1">Col #1</entry>
<entry valign="top" rowsep="1">Col #2</entry>
<entry valign="top" rowsep="1">Col #3</entry>
<entry valign="top" rowsep="1">Col #4</entry>
</row>
</thead>
```

```
<tbody>
<row>
<entry valign="top">Row 1, Cell 1</entry>
<entry valign="top"><ol>
<li>Row 1</li>
<li>Cell 2</li>
</ol></entry>
<entry valign="top">Row 1, Cell 3; here's a little
more text than the other cells have</entry>
<entry valign="top">Row 1, Cell 4</entry>
</row>
<row>
<entry valign="top">Row 2, Cell 1</entry>
<entry valign="top">Row 2, Cell 2</entry>
<entry valign="top"><ph style="italic">Row 2, Cell
3</ph></entry>
<entry valign="top">Row 2, Cell 4</entry>
</row>
</tbody>
</tgroup>
</table>
```

Attributes

VALIGN=TOP | MIDDLE | BOTTOM
This attribute specifies the vertical alignment of the text contained in the Entry element.

TOP
specifies alignment of the text at the top of the Entry.

MIDDLE
specifies alignment of the text at the middle of the Entry.

BOTTOM
specifies alignment of the text at the bottom of the Entry (the default).

Usage

See “A Simple Table with a Table Header and IBMIDDoc Elements” on page 78.

Contexts

Children: row.

Parents: tgroup.

Then (procedure action to take)

Purpose

The Then element contains a description of the action to take as the result of a condition that occurs at a decision point of a procedure.

Examples

```
<decisionpnt>
<cond>Is the diaper wet?</cond>
<then><procstep><proccmnd>
<desc>Change the diaper.</desc>
</proccmnd><procexit>Johnny was uncomfortable.</procexit>
</procstep>
</then>
```

Then (procedure action to take)

```
<else>
<desc>Continue at <xref refid="hungry">.</desc>
</else>
</decisionpnt>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 22, “Creating maintenance analysis procedures” on page 207.

Contexts

Children: Desc, Proc, ProcStep.

Parents: DecisionPnt.

Title

Purpose

The Title element contains a title for elements that can have titles. Note that the meaning of the Title element is dependent upon the context in which it is used. For example, when used within a Division, Title contains the chapter or topic heading. When used in a PBlk element, it contains the title for the block of paragraphs.

Examples

```
<d>
<dprolog><titleblk>
<title>My Little Chapter</title>
</titleblk></dprolog>
<dbody>
<p>Here's the beginning of my chapter.</p>
</dbody></d>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Creating divisions (D element)” on page 20.

Contexts

Children: text (#pcdata), L, Ph, Term, TM.

Parents: Annot, Author, Bridge, ClassDef, DLBlk, EdNotices, Fragment, FragRef, GLBlk, Group, LIBlk, ModInfo, ModItemDef, Msg, MsgItemDef, Note, NoteList, ParmBlk, PartAsm, PBlk, PropDesc, Qualif, SynBlk, Syntax, TitleBlk.

TitleBlk (title information)

Purpose

The TitleBlk element contains title information.

Examples

```
<d>
<dprolog><titleblk>
<title>My Little Chapter</title>
</titleblk></dprolog>
<dbody>
<p>Here's the beginning of my chapter.</p>
</dbody></d>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Creating divisions (D element)” on page 20.

Contexts

Children: STitle, SubTitle, Title.

Parents: DIntro, DocTitle, DProlog, DSum, FigList, IBMSafety, Index, Library, PNIndex, Proc, ProcStep, RCF, Safety, SpecDProlog, TList, TOC.

TList (list of tables)

Purpose

The TList element either causes a table list to be generated, or contains an explicit list of tables to be presented.

Examples

```
<tlist><gendtitle></tlist>
```

Attributes

SPEC=AUTO | MAN
Specifies whether the content of the element is generated. If SPEC=AUTO is specified, the list is automatically generated.

| **LAYOUT=Default-Layout | OneCol | TwoCol**
| Specifies the column-style for the section.

| **Default-Layout**
| The section uses the default layout for the document style.

| **OneCol**
| The headings and text format across the entire page.

| **TwoCol**
| The text formats in two columns. Headings format across the page or
| with the two-column text.

See “Common Element Attributes (large set)” on page 227.

Usage

See “List of tables” on page 100.

TList (list of tables)

Contexts

Children: CLE, GendTitle, RetKey, TitleBlk.

Parents: FrontM.

TM (Trademark)

Purpose

The TM tag identifies the trademark terms in the document.

The TM tag identifies the trademark terms in your source by surrounding the trademark term or phrase. This tag has attributes that are not translated; they contain no “MRI”. The TM attributes contain information for the author an the processing formatters. The TMType attribute creates the appropriate trademarking character after the term or phrase. The files IDDIRTM.LST or IDTMSCAN.LST list the trademarks and the attributes needed for the TM tag. Use these files to insert the TM tag with proper attributes for the trademark term or phrase.

Examples

```
<tm trademark="OS/2" tmowner="IBM Corporation" tmtype="reg"
tmclass="ibm">OS/2</tm> requires a <TM trademark="Pentium"
tmowner="Intel Corproation" tmtype="reg" tmclass="special">Pentium</tm>
166MHz processer.
```

Attributes

trademark
This is the trademark term or phrase repeated in the tag. This attribute is required.

tmowner
This identifies the trademark owner. For example, the trademark owner could be “IBM Corporation”. This attribute is optional.

tmtype
This identifies the trademark type. One of the following must be chosen:

TM
Trademark™.

REG
Registered trademark®.

SERVICE
Service marks^{SM}.

tmclass
This identifies the trademark classification. One of the following must be chosen:

IBM
IBM Corporation.

IBMSUB
IBM subsdiary (such as Tivoli™ or Lotus).

SPECIAL
Requires special notation. These are for companies who have a special

agreement with IBM. There is a legal obligation for IBM to mark these trademarks in the document as well as in the Notices section.

OTHER
All other trademarks. These are not marked in the output.

Usage

See the *ID Workbench Getting Started and User's Guide* for the tools to insert trademarks into your document.

Contexts

Children: text (#pcdata), Ph, Term.

Parents: Address, AnnotBody, Attention, Bridge, Cap, Caution, CI, CLE, CompCmt, Cond, Copyr, Danger, Defn, DefnHd, Desc, entry, Fn, L, LEDesc, LEN, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, ModName, MsgText, NoteBody, P, Ph, ProcEntry, ProcExit, Q, Sem, STitle, SubTitle, SynNote, Term, TermHd, Title, Warning.

TOC (table of contents)

Purpose

The TOC element either causes a table of contents to be generated, or contains an explicit list of CLE elements to be presented.

Examples

```
<toc><gendtitle></toc>
```

Attributes

SPEC=AUTO | MAN
Specifies whether the content of the element is generated. If SPEC=AUTO is specified, then the element's content is generated.

| **LAYOUT=Default-Layout | OneCol | TwoCol**
| Specifies the column-style for the section.

| **Default-Layout**
| The section uses the default layout for the document style.

| **OneCol**
| The headings and text format across the entire page.

| **TwoCol**
| The text formats in two columns. Headings format across the page or
| with the two-column text.

See "Common Element Attributes (large set)" on page 227.

Usage

See "Table of contents" on page 99.

TOC (table of contents)

Contexts

Children: CLE, GendTitle, RetKey, TitleBlk.

Parents: DIntro, FrontM.

UL (unordered list)

Purpose

The UL element contains a list of items whose order of appearance is not important.

Migration Note

The simple list element from BookMaster has been included into the unordered list element; the only difference is the type of bullet used for the two lists.

Examples

```
<ul>
<li>This is an item in an unordered list. To separate
it from other items in the list, the formatter puts
a bullet beside it.</li>
<li>The paragraph that is contained in the LI element
is part of the list item which contains it. <p>This
is the contained paragraph.</p></li>
<li>This is a separate list item in our unordered
list.</li>
</ul>
```

Attributes

| **ULTYPE= checkoff | <u>normal</u> | simple | simplecheckoff**
| Specifies the type of the list. Checkoff lists have an underscore before the
| bullet. Simple lists have no bullet or dash before the list item. Simplecheckoff
| lists have only the underscore in front of the list item.

style= simple
Deprecated. Use ULTYPE=SIMPLE. Specifies a simple list; no bullet is produced.

| **DBScalePct=***scale-percent*
| You can use the DBScalePct attribute to scale the dingbat (the thing in front of
| the list) up or down for a list item. The scale-percent is one of the following
| numbers: 50, 60, 70, 80, 90, 100, 110, 120, 140, 160, 180, 200, 240, or 300. 100 is
| the normal size; 50 is 50% smaller; 200 is 200% larger. For example, this scales
| the bullet or number to twice its size:

```
dbscalepct="200"
```

| This works for hardcopy only.

LINESPACE=SPACE | COMPACT
Specifies whether the items in the list should be compacted or have space between the items. Nested lists automatically inherit the setting of the outer list, but can override that default with their own setting.

DEF=*definition-name*
: Specifies the definition to be used. This points to a DefName attribute on a corresponding DEF tag. The settings defined on that DEF tag are used as defaults for this tag. See Chapter 9, “Using definition tags” on page 105 for more information.

Usage

See “Unordered lists” on page 29.

Contexts

Children: Bridge, LI, LIBlk.

Parents: AnnotBody, Attention, BackCover, Bridge, Caution, Cond, Copyr, Danger, DBody, Defn, Desc, DIntro, DSum, EdNotices, entry, Fig, FigSeg, Fn, FrontCover, LEDesc, LEDI, LI, LQ, MkNote, ModDesc, ModItem, MsgItem, NItem, NoteBody, Notices, P, PBlk, ProcEntry, ProcExit, ProcIntro, Safety, Sem, SynNote, TextAlt, Warning.

ULDef (Unordered list definition)

Purpose

The ULDef element sets attribute defaults for unordered lists. ULDef goes within the document prolog to set definitions for the entire document; or within a division prolog to set definitions for just that division. The ULDef tag goes inside a PropDefs tag.

Examples

```
<propdefs><uldef defname="checklist" ultype="checkoff"
dbscalepct="140"></propdefs>
...
<ul def="checklist">
<li>old horse</li>
<li>kitty-cat</li>
<li>peanut butter</li>
<li>engine</li>
</ul>
```

Attributes

DBScalePct=*scale-percent*
: You can use the DBScalePct attribute to scale the dingbat (the thing in front of the list) up or down for a list item. The scale-percent is one of the following numbers: 50, 60, 70, 80, 90, 100, 110, 120, 140, 160, 180, 200, 240, or 300. 100 is the normal size; 50 is 50% smaller; 200 is 200% larger. For example, this scales the bullet or number to twice its size:

```
dbscalepct="200"
```

This works for hardcopy only.

DEFNAME=*definition-name*
: The DefName attribute identifies a definition element. DefName values must be unique within a single document. A element that has a DEF attribute can point a corresponding tag with a DefName attribute. DefName values can be up to 64 characters long. DefName values must start with an alphabetic character and can contain letters, numbers, dashes (-), or periods (.).

Definition tags can be specified without a DefName attribute in the document's prolog. This allows you to change the initial settings of attributes for the entire document when the DEF tag is in the document's prolog. In a division's prolog, it changes the initial settings for that division.

LINESPACE=SPACE | COMPACT
Specifies whether the items in the list should be compacted or have space between the items. Nested lists automatically inherit the setting of the outer list, but can override that default with their own setting.

ULTYPE= checkoff | normal | simple | simplecheckoff
Specifies the type of the list. Checkoff lists have an underscore before the bullet. Simple lists have no bullet or dash before the list item. Simplecheckoff lists have only the underscore in front of the list item.

Props=*properties*
The Props attribute is used to specify the condition under which the information contained within the element appears. See "Property-Based Retrieval" on page 195.

Usage

See Chapter 9, "Using definition tags" on page 105.

Contexts

Children: empty.

Parents: PropDefs, PropGroup.

Var (syntax variable)

Purpose

Use Var to define variables within a syntax definition.

Examples

```
<syntax>
<group>
<kwd>LANGUAGE</kwd>
<oper>=</oper>
<var>language_name</var>
</group>
</syntax>
```

Attributes

OPTREQ=OPT | REQ | DEF
Indicates whether or not the variable is optional.

LINKEND=*id*
Contains an ID reference that enables a link to an arbitrary location in the document.

See "Common Element Attributes (large set)" on page 227.

Usage

See Chapter 15, "Programming Syntax Diagrams" on page 147.

Contexts

Children: text (#pcdata).

Parents: Group, SynPh.

Version (product version number)

Purpose

The Version element contains the version number of the product that the document describes.

Examples

```
<ibmprodinfo>
<prodname>ID Workbench</prodname>
<version>Version 37</version>
<release>Release 29</release>
</ibmprodinfo>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “Other prolog elements” on page 90.

Contexts

Children: text (#pcdata), Ph.

Parents: IBMProdInfo, ProdInfo.

VNet (IBM VNet mail address)

Purpose

The VNet element contains an IBM VNet email address.

Examples

```
<maintainer>
<corp>
<corpname>IBM Corporation</corpname>
<address>ATTN: Dept 542
3605 HWY 52 N
Rochester, MN
<postalcode>55901-9986</postalcode><phone equip="fax">
1-800-555-1212</phone><vnet>http://w3.rchland.ibm.com/projects/IDWB
</vnet></address>
</corp>
</maintainer>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

VNet (IBM VNet mail address)

Contexts

Children: text (#pcdata).

Parents: Address.

Volid (volume identifier)

Purpose

Contains the identifier for one portion of a multivolume document.

Examples

```
<doctitle>
<library><titleblk>
<title>ID Workbench</title>
</titleblk></library>
<titleblk>
<title>IBMIDDoc User's Guide <ph props="ref">and
Reference</ph></title>
</titleblk></doctitle><volid>Volume 1</volid><ibmdocnum>
SH21-0783-09</ibmdocnum><externalfilename>iddugref
</externalfilename>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Contexts

Children: text (#pcdata), Ph.

Parents: IBMBibEntry.

Warning (warning notice)

Purpose

Use Warning to contain a mandatory safety notice, consisting of one or more paragraphs or other paragraph-level elements. For non-mandatory notices, use the Attention element. See “Attention (safety notice)” on page 238 for information about the Attention element.

Examples

```
<WARNING>Do not use this blow dryer while taking a shower.
</WARNING>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Usage

See “The perils of processing: Attention, caution, and danger” on page 48.

Contexts

Children: text (#pcdata), Address, Annot, APL, Bin, Bridge, CGraphic, Char, Cit, Date, Dec, DL, Fig, Formula, GL, Hex, L, Lines, LitData, LQ, MD, MkNote, MMObj, ModInfo, MV, Num, Oct, OL, P, ParmL, PBlk, Ph, PK, PV, Q, RefKey, Screen, SynPh, Syntax, Table, Term, TM, UL, Xmp, XPh, XRef.

Parents: Safety.

WebPage

Purpose

Use WebPage to contain a web-page address. Use this to provide a location in your book to allow others to “read more about it”. Webpage is used as part of an address. It currently does not cause any automated linking from a PDF file. When used in the document, it is printed as an address line. When used in the Maintainer section, it is printed on non-US reader comment forms.

Examples

```
<maintainer>
<corp>
<corpname>IBM Corporation</corpname>
<address>ATTN: Dept 542
3605 HWY 52 N
Rochester, MN
<postalcode>55901-9986</postalcode><phone equip="fax">
1-800-555-1212</phone>
<webpage>http://w3.rchland.ibm.com/projects/IDWB</webpage>
</address>
</corp>
</maintainer>
```

Attributes

See “Common Element Attributes (large set)” on page 227.

Contexts

Children: text (#pcdata).

Parents: Address.

Xmp (example)

Purpose

Use Xmp to contain examples of computer input or output, such as code samples or listings. In the default style, Xmp data is presented in a monospaced typeface. Within Xmp, significant record ends are preserved and presented.

Examples

```
<XMP STYLE='BKM:(KEEP="10")'>
10 LET A = B
20 IF A GT C THEN GO 40
30 LET A = C
40 PRINT A, C
</XMP>
```

Attributes

OBJ=object-reference
Allows you to include a declared program entity.

NOTATION=LINESPEC
LINESPEC is the default value for the NOTATION attribute on Xmp.

LINELENGTH=*characters*
This specifies the number of characters in widest line. If the number of characters does not fit in the column or page width, the text size is reduced.

PGWIDE=1 | 2
This specifies the width of the example. 1 is for a page-wide example; 2 indents to the current indention in the column (2 is the default).

DEF=*definition-name*
Specifies the definition to be used. This points to a DefName attribute on a corresponding DEF tag. The settings defined on that DEF tag are used as defaults for this tag. See Chapter 9, "Using definition tags" on page 105 for more information.

See "Common Element Attributes (large set)" on page 227.

Usage

See "Examples of computer output" on page 54.

Contexts

Children: text (#pcdata), L, LitData, MV, Ph, PK, PV, RefKey, SynPh, Term.

Parents: AnnotBody, Attention, BackCover, Bridge, Caution, Danger, DBody, Defn, DIntro, DSum, entry, Fig, FigSeg, Fn, FrontCover, LEDI, LI, LQ, MkNote, ModDesc, ModItem, MsgItem, NoteBody, P, PBlk, ProcIntro, SynNote, Warning.

XmpDef (Example definition)

Purpose

The XmpDef element sets attribute defaults for exampless. XmpDef goes within the document prolog to set definitions for the entire document; or within a division prolog to set definitions for just that division. The XmpDef tag goes inside a PropDefs tag.

Examples

```
<xmpdef defname="widexmp" linelength="132">
```

Attributes

DEFNAME=*definition-name*
The DefName attribute identifies a definition element. DefName values must be unique within a single document. A element that has a DEF attribute can point a corresponding tag with a DefName attribute. DefName values can be up to 64 characters long. DefName values must start with an alphabetic character and can contain letters, numbers, dashes (-), or periods (.).

Definition tags can be specified without a DefName attribute in the document's prolog. This allows you to change the initial settings of attributes for the entire document when the DEF tag is in the document's prolog. In a division's prolog, it changes the initial settings for that division.

PGWIDE=1 | 2
This specifies the width of the example. 1 is for a page-wide example; 2 indents to the current indention in the column (2 is the default).

LINELENGTH=*characters*
This specifies the number of characters in widest line. If the number of characters does not fit in the column or page width, the text size is reduced.

Props=*properties*
The Props attribute is used to specify the condition under which the information contained within the element appears. See "Property-Based Retrieval" on page 195.

Usage

See Chapter 9, "Using definition tags" on page 105.

Contexts

Children: empty.

Parents: PropDefs, PropGroup.

XPh (example phrase)

Purpose

Use the XPh element for computer input or output phrase that occurs within text. In the default style, XPh is presented in the same typeface as is used for Xmp elements.

Examples

```
The system will respond with a <XPH>READY</XPH> message.
```

Attributes

See "Common Element Attributes (large set)" on page 227.

Usage

See Table 1 on page 44.

Contexts

Children: text (#pcdata), L, MV, Ph, PK, PV, Term.

Parents: AnnotBody, Attention, Bridge, Caution, CompCmt, Danger, Defn, Desc, entry, Fn, L, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, NoteBody, P, Ph, Q, SynNote, Term, Warning.

XRef (cross reference)

Purpose

The XRef element defines a reference to another element and generates the reference text (or other link indicator) automatically. The element referred to can be in the same document or in another document. XRef can point to another element indirectly by referring to a NameLoc element.

XRef defines a link to another element and automatically generates the link indicator. The link indicator is normally the title of the element linked to along with some locator, such as a page or panel reference, or, when the element does not have a title, just a locator. When the element linked to has an XRefText attribute specified, the XRefText value is used as the link indicator. For example, a cross reference to a paragraph that does not have XRefText coded would generate just a page number.

The style of the generated cross reference is determined by the active document style.

You can create a cross reference indirectly by using a NameLoc element to define the target of the cross reference. For example, you may have already defined a NameLoc to something for use by the L element because it will be linked to many times. You can use this same NameLoc for cross references to the element.

When XRef references a NameLoc element which references another document using the DOCNAME attribute, a cross-document link is generated.

The name on the NameLoc's DOCNAME attribute must match the name specified on the DOCNAME attribute on an IBMBibEntry or BibEntry element. DOCNAME values must be unique for each IBMBibEntry or BibEntry element.

Note: XREFTEXT is the only IBMIDDoc attribute that can take DBCS data.

Examples

Referring directly to another element:

```
<P>See <XREF REFID="AboutXRef"> for more information.
 ...
<D ID="AboutXref">XRef Explained
```

Here is an example referring to a title using Form=Full:

```
<title id=heading>Basic use of Elements</title>
 ...
<p>The best place for more information is in
<xref refid=heading form=full>.</p>
```

The example would read:

```
The best place for more information is in
"Chapter 3. Basic use of Elements" on page 20.
```

Here is an example referring to an ordered list using Form=Full:

```
<p>Here is a list of important items:
  <ol>
     <li>This is the first important item</li>
     <li>This is the second important item</li>
     <li id=third>This is the third important item</li>
  </ol></p>
<p>This information refers back to the important list item
<xref refid=third form=full>
```

Which generates this output:

```
This information refers back to the important list item 3 on
page 5.
```

Here is an example referring to a title using Form=Text:

```
<title id=Heading>Basic Use of Elements</title>
   ⋮
<p>The best place for more information is in
<xref refid=heading form=text>.
```

The example shows:

```
The best place for more information is in Chapter 3. Basic Use of
Elements.
```

Attributes

REFID=*element_id*
The ID value of the element being linked to. The linked element may be any element that takes an ID attribute.

OBJTYPE=*target_type*
Specifies the target type of the object being referenced.

FORM=*formtype*
There are three types of cross references that can be used with the Form attribute: text, number, or location. Text elements are usually linked to titles. The location is a page number for printed output. The number is for enumerated elements only (for example, ordered list items). Be sure to check out the examples section to see how some of the attributes are used. If you are processing the document through a transform that generates a hard copy version, you can control the form of the reference. If you are processing the document through a transform that generates an online version, the xref form attribute is currently ignored. Here are the specific choices listed under Form:

NORMAL
The active presentation style defines the output.

FULL
A full reference includes all variable information about the target element. For example, if you were referring to a chapter heading, the xref would list the chapter number, name of the chapter, and the page number where the chapter can be found.

TEXT
This lists the text of the target element (typically the title). This will list the chapter number and the name of the chapter. The page number is not listed.

NUMBER
This lists only the number of the target. If this was used in reference to a chapter title, only the chapter number would be listed. If this was used in reference to an ordered list, the xref would only list the number of the step--not the page where the list number is located.

LOCATION
The location of an element, such as its page number or panel ID. In some presentations, there may not be a meaningful locator, in which case the presentation style may do the best it can, such as presenting the title of the nearest containing element.

These renamining attributes may be supported in the future; they currently processes the same as Normal.

NUMLOC
The number of an enumerated element followed by its location.

NUMTEXT
The number of an enumerated element followed by its text.

LOCTEXT
The location of an element followed by its text.

TEXTLOC
The text of an element followed by its location.

TEXTNUM
The text of an element followed by its number.

LOCNUM
The location of an element followed by its number.

See “Common Element Attributes (large set)” on page 227.

Usage

See Chapter 6, “Cross-referencing” on page 61.

Contexts

Children: empty.

Parents: AnnotBody, Attention, Bridge, Caution, CompCmt, Danger, Defn, Desc, entry, Fn, Group, L, LI, Lines, LQ, MD, MkNote, ModDesc, ModItem, NoteBody, P, Ph, Q, SynNote, Warning.

Part 4. Appendixes

Appendix A. IBMIDDoc Supported Notations

The following table lists the notations and their identifiers.

Table 24. Notation table

Notation name	Notation identifier
SGMLDOC	PUBLIC ″ISO 8879:1986//NOTATION STANDARD GENERALIZED MARKUP LANGUAGE (SGML)//EN″
SMFF	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION SCRIPT MATHEMATICAL FORMULA FORMATTER//EN″
PSEG	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION AFP PAGE SEGMENT//EN″
PSEG3820	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION AFP PAGE SEGMENT::3820//EN″
PSEG38PP	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION AFP PAGE SEGMENT::38PP//EN″
PSEG4250	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION AFP PAGE SEGMENT::4250//EN″
ANIMATION	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION GENERIC ANIMATION META-NOTATION//EN″
VIDEO	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION GENERIC VIDEO META-NOTATION//EN″
AUDIO	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION GENERIC AUDIO META-NOTATION//EN″
GRAPHICS	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION GENERIC GRAPHICS META-NOTATION//EN″
VECTOR	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION GENERIC VECTOR META-NOTATION//EN″
IMAGE	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION GENERIC IMAGE META-NOTATION//EN″
EPS	PUBLIC ″-//ADOBE//NOTATION ENCAPULATED POSTSCRIPT//EN″
APL	PUBLIC ″ISO 8485:1989//NOTATION PROGRAMMING LANGUAGES - APL//EN″
ASM	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION 80X86 ASSEMBLER PROGRAMMING LANGUAGE//EN″
ASSEMBLE	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION 370 ASSEMBLER PROGRAMMING LANGUAGE//EN″
BAT	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION DOS BAT PROGRAMMING LANGUAGE//EN″
C	PUBLIC ″ISO/IEC 9899:1990//NOTATION PROGRAMMING LANGUAGES - C//EN″
COBOL	PUBLIC ″ISO 1989:1985//NOTATION PROGRAMMING LANGUAGES - COBOL//EN″
CPP	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION C++ PROGRAMMING LANGUAGE//EN″
EXEC	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION EXEC PROGRAMMING LANGUAGE//EN″
FORTRAN	PUBLIC ″ISO/IEC 1539:1991//NOTATION INFORMATION TECHNOLOGY - PROGRAMMING LANGUAGES - FORTRAN//EN″
PLI	PUBLIC ″ISO 6160:1979//NOTATION PROGRAMMING LANGUAGES - PL/1//EN″

Table 24. Notation table (continued)

Notation name	**Notation identifier**
REXX	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION REXX PROGRAMMING LANGUAGE//EN″
CMNDLINE	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION COMMAND LINE ENTRY//EN″
HYQ	PUBLIC ″ISO/IEC 10744:1992//NOTATION HYTIME QUERY NOTATION//EN″
HYLEX	PUBLIC ″ISO/IEC 10744:1992//NOTATION HYTIME LEXICAL MODEL NOTATION//EN″
SCRIPT	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION DOCUMENT COMPOSITION FACILITY MARKUP//EN″
SCREEN	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION CHARACTER SCREEN REPRESENTATION//EN″
LINESPEC	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION LINE SPECIFIC CONTENT//EN″
PROGRAM	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION GENERIC PROGRAM LAUNCH META-NOTATION//EN″
BOOKMANAGERBOOK	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION BOOKMANAGER COMPILED BOOK NOTATION//EN″
IPFINF	PUBLIC ″+//ISBN 0-933186::IBM//NOTATION INFORMATION PRESENTATION FACILITY BOOK NOTATION//EN″
URL	PUBLIC ″-//IETF //NOTATION UNIVERSAL RESOURCE LOCATOR NOTATION:: IETF RFC 1738//EN″

Appendix B. Proposed IBM Standard for Formal Public Identifiers

One of the SGML mechanisms on which IBMIDDoc depends is Formal Public Identifiers (FPIs). This is a system-independent naming syntax that is defined in the standards ISO 8879 and ISO 9070. There are several advantages to using them, as opposed to using system identifiers directly. Resolution of references to these entities are supported using a catalog lookup feature found in most SGML products. This paper describes the conventions to be used within IBM when using these formal public identifiers.

Here is the basic format of a formal public identifier:

FPI Format

```
►►──┬─+─┬──//owner identifier──//public text class── public text description─────►
    └─-─┘

►─//public text language────────────────────────────────────────────────────────►◄
```

The following sections will describe each of these fields. In addition, these sections describe a format to be used within IBM information development when creating formal public identifiers. Identifiers which comply with this standard will be unique and consistent across the corporation. This may become a the standard for other parts of the corporation. Our customers may wish to use a definition like this in their own work.

Notice that when processed, FPIs are transformed or normalized using the same normalization rules as those used for tokenizing attribute value literals:

1. Ignore RS
2. Replace RE and SEPCHAR with SPACE
3. Replace a sequence of SPACE characters with a single SPACE and
4. Ignore leading and trailing SPACE characters.

This means that record ends or multiple spaces may be freely inserted anywhere a space is allowed.

Owner Identifier

The first part described here is a structure called an owner identifier. There are two forms of interest: registered (preceded with a "+") and unregistered (preceded with a "-"). The ISO 9070 standard describes a registration process and an authority. It also defines a syntax for using a registered identifier based on an ISBN number. Either the 9070 registration or the ISBN registration may be used in registered owner identifiers. Since the registration authority has not yet been named, the ISBN number syntax must be used.

IBM has an ISBN publisher prefix, 0-933186. This will be used for all formal public identifiers for IBM-owned objects. This yields a registered owner identifier of `+//ISBN 0-933186::IBM`.

Public Text Class and Public Text Description

Following the owner identifier, there are two fields called the public text class and public text description. The public text class defines the type or class of entity being named. The public text description gives more information about the entity described.

Here is an example of a formal public identifier which conforms to this standard.

```
+//ISBN 0-933186::IBM//DOCUMENT PUB SC31-1234-00//EN
```

The "+//ISBN ..." is the registered owner identifier, in this case indicating that the owner is IBM using the ISBN registration. Notice that **0-933186** represents the ISBN registration identifier that has been assigned.

The keyword **DOCUMENT** defines the public text class for the entity. The list of possible values for this field are defined in ISO 8879. It is also reiterated in the list below. The keyword **DOCUMENT** indicates that the entity identified is a complete SGML document.

The keyword **PUB** is the start of the public text description. The keyword **PUB** in this context indicates an IBM publication and is followed by the standard IBM publication number. The "EN" preceded by the "//" is the public text language identifier, in this case, English.

The following list defines the public text description field format. Notice that its value is keyed on the public text class. Unless otherwise noted, the data in the field identifies the entity itself; in some cases it may refer to either a containing context or another entity to which the defined information is related. As used in these definitions, the term “context” means the document library or collection to which the entity applies or which *owns* the entity.

Public Text Class
  Public Text Description

CAPACITY

  The following field identifies the DTD to which this capacity set entity applies.

  Corporate Standard
    STD ppp n-nnnn-nnn [nnnn-n]

    A corporate standard will be written for all IBM internal DTDs which are distributed throughout the company. The CAPACITY file may be used to maintain the CAPACITY used by these DTDs and is referenced from SGML declaration files which apply to the DTD.

  Local Definition
    LCL owner descriptor [::descriptor]

CHARSET
  [CP nnnnn[-n]]

  where

nnnnn[-n]
: indicates a standard IBM codepage, if specified. If not specified, the character set is assumed to not be an IBM-defined codepage.

Notice that if another standard is used for character encoding, such as an ANSI or ISO standard, the FPI for their standards, as they define them, should be used.

DOCUMENT

The following field applies to the document being described, not an owning context.

Corporate Standard
: **STD** ppp n-nnnn-nnn [nnnn-n]

IBM publication
: **PUB** zznn-nnnn[-nn]

IBM collection
: **LIB** zznn-nnnn[-nn]

ISBN notation
: **ISBN** n-nn-nnnnnn-n

Local Definition
: **LCL** owner descriptor [::descriptor]

DTD

Corporate Standard
: **STD** ppp n-nnnn-nnn [nnnn-n]

A corporate standard will be written for all IBM internal DTDs which are distributed throughout the company.

Local Definition
: **LCL** owner descriptor [::descriptor]

Notice that if a standard DTD is used from another institution, like ANSI or ISO, the FPI for their standards, as they define them, should be used.

ELEMENTS

If the declared elements are intended for general use throughout IBM, the entity would be identified by the corporate standard identifier of the DTD to which the declarations are related.

Corporate Standard
: **STD** ppp n-nnnn-nnn [nnnn-n][::descriptor]

If elements are defined for a particular document, library or other collection, the identifier identifies the owning context.

IBM publication
: **PUB** zznn-nnnn[-nn][::descriptor]

IBM collection
: **LIB** zznn-nnnn[-nn][::descriptor]

ISBN notation
: **ISBN** n-nn-nnnnnn-n[::descriptor]

Local Definition
: **LCL** owner descriptor [::descriptor]

ENTITIES

When an entity set is distributed throughout the company, a corporate standard will be written for it and it will be identified using this field:

Corporate Standard
: **STD** ppp n-nnnn-nnn [nnnn-n]

When an entity set is used to hold the entity declarations for a particular document, library or collection, the identifier for that context should be used to identify the entity:

IBM publication
: **PUB** zznn-nnnn[-nn][::descriptor]

IBM collection
: **LIB** zznn-nnnn[-nn][::descriptor]

ISBN notation
: **ISBN** n-nn-nnnnnn-n[::descriptor]

Local Definition
: **LCL** owner descriptor [::descriptor]

Notice that if a standard entity set is used from another institution, like American Mathematical Society or ISO, the FPI for their standards, as they define them should be used.

LPD

When a link process declaration set is distributed throughout the company, a corporate standard will be written for it and it should be identified using this field:

Corporate Standard
: **STD** ppp n-nnnn-nnn [nnnn-n]

When an LPD is used to hold the declarations for a particular document, library or collection, the identifier for that context should be used to identify the entity:

IBM publication
: **PUB** zznn-nnnn[-nn][::descriptor]

IBM collection
: **LIB** zznn-nnnn[-nn][::descriptor]

ISBN notation
: **ISBN** n-nn-nnnnnn-n[::descriptor]

Local Definition
: **LCL** owner descriptor [::descriptor]

Note: The LINK features are not currently used within IBM but this specification is included for completeness and in anticipation of when these features will be used.

NONSGML

The following fields identify the owning context for the non-SGML entity.

Corporate Standard
: **STD** ppp n-nnnn-nnn [nnnn-n][::descriptor]

IBM publication
: **PUB** zznn-nnnn[-nn][::descriptor]

IBM collection
: **LIB** zznn-nnnn[-nn][::descriptor]

ISBN notation
: **ISBN** n-nn-nnnnnn-n[::descriptor]

Local Definition
: **LCL** owner descriptor [::descriptor]

NOTATION

The following field identifies the document which defines the notation referred to by this notation identifier.

Corporate Standard
: **STD** ppp n-nnnn-nnn [nnnn-n]

Local Definition
: **LCL** owner descriptor [::descriptor]

Notice that if a notation is defined by a standards body, like ANSI or ISO, the FPI for their standards, as they define them should be used.

SHORTREF

When a short reference declaration set is distributed throughout the company, a corporate standard will be written for it and it should be identified using this field:

Corporate Standard
: **STD** ppp n-nnnn-nnn [nnnn-n]

When a short reference set is used to hold the declarations for a particular document, library or collection, the identifier for that context should be used to identify the entity:

IBM publication
: **PUB** zznn-nnnn[-nn]

IBM collection
: **LIB** zznn-nnnn[-nn]

ISBN notation
: **ISBN** n-nn-nnnnnn-n

Local Definition
: **LCL** owner descriptor [::descriptor]

Note: The short reference capabilities of SGML are not currently used by IBMIDDoc. This section is included for completeness. There are no plans to use short references with IBMIDDoc within IBM.

SUBDOC

The following field applies to the document being described, not an owning context.

Corporate Standard
: **STD** ppp n-nnnn-nnn [nnnn-n]

IBM publication
: **PUB** zznn-nnnn[-nn]

ISBN notation
: **ISBN** n-nn-nnnnnn-n

Local Definition
: **LCL** owner descriptor [::descriptor]

SYNTAX

The following field identifies the DTD to which this syntax definition entity applies.

Corporate Standard
: **STD** ppp n-nnnn-nnn [nnnn-n]

A corporate standard will be written for all IBM internal DTDs which are distributed throughout the company. The SYNTAX file may be used to maintain the SYNTAX used by these DTDs and is referenced from SGML declaration files which apply to the DTD.

When an syntax definition only applies to a particular document, library or collection, the identifier for that context should be used to identify the entity:

IBM publication
: **PUB** zznn-nnnn[-nn]

IBM collection
: **LIB** zznn-nnnn[-nn]

ISBN notation
: **ISBN** n-nn-nnnnnn-n

Local Definition
: **LCL** owner descriptor [::descriptor]

TEXT

The following fields identify the owning context for the SGML text entity.

Corporate Standard
: **STD** ppp n-nnnn-nnn [nnnn-n][::text descriptor]

IBM publication
: **PUB** zznn-nnnn[-nn][::text descriptor]

IBM collection
: **LIB** zznn-nnnn[-nn][::text descriptor]

ISBN notation
: **ISBN** n-nn-nnnnnn-n[::text descriptor]

Local Definition
: **LCL** owner descriptor [::descriptor]

Where

author name
: This is a name which uniquely defines the author. It may be an RSCS address, an Internet address, an employee serial number. The key is that it must be unique across the corporation.

descriptor
: is a unique identifier within the scope of the owning context that identifies that content.

owner descriptor
identifies the owner of the data. This need not be unique; it could be the name of the author, the owning department or site.

nnnnn[-n]
This is the identifier for the IBM codepage being specified.

ISBN n-nn-nnnnnn-n
This is an ISBN number.

STD ppp n-nnnn-nnn [nnnn-n]
This is the identifier defined for the specific IBM corporate standard in question.

text descriptor
is a unique identifier within the scope of the owning context. It has the following format:

```
descriptor[/author name[/owner descriptor]]
```

PUB or LIB zznn-nnnn[-nn]
This is the identifier of a document or library.

Public Text Language

The public text language field indicates the language used within the entity identified by the formal public identifier. The language codes which are valid for this field are defined in ISO 639.

Appendix C. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created

Trademarks

Part Number Index

Part Number	Asm–Index	Page
1230987	1–1	216
1234939	1–3	216
1238475	1–2	216
33-5234	2–1	219
4563423	1–1	216
56-2345	2–	219
56-3476	2–5	219
56-3489	2–3	219
56-4352	2–2	219
56-6534	2–4	219
56-8393	2–6	219
56-9845	2–7	219
56-9874	2–8	219

Index

Readers' Comments — We'd Like to Hear from You

ID Workbench
IBMIDDoc User's Guide and Reference
Release 3.6

Publication No. SH21-0783-10

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name	Address
Company or Organization	
Phone No.	

Readers' Comments — We'd Like to Hear from You
SH21-0783-10

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
ATTN: Dept 542
3605 HWY 52 N
Rochester, MN
55901-9986

Fold and Tape **Please do not staple** Fold and Tape

SH21-0783-10

Cut or Fold
Along Line

Printed in U.S.A.

SH21-0783-10