

CARNEGIE-MELLON UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

SPICE PROJECT

**User Manual for Mint —
The Spice Document Preparation System**

Peter Hibbard

3 April 83

Abstract

This document describes version 2A(21) of Mint, the Spice Document Formatter. This is an early draft, and not all the facilities of Mint are accurately described. The whole of the document has been produced by Mint and DP, executing on a Perq under POS.

Spice Document S153

Keywords and index categories: none

Location of machine-readable file: refman.mss

Copyright © 1983 Peter Hibbard

This is an internal working document of the Computer Science Department, Carnegie-Mellon University, Schenley Park, Pittsburgh, PA 15213. Some of the ideas expressed in this document may be only partially developed, or may be erroneous. Distribution of this document outside the immediate working community is discouraged; publication of this document is forbidden.

Table of Contents

1 Getting Started	1
1.1 Mint and Scribe	1
1.2 Getting Started	2
1.2.1 Loading Mint onto a Perq	2
1.2.2 Running Mint	2
1.3 What to do in case of trouble	4
1.3.1 Reporting Errors	4
1.4 The Future	5
2 Design Philosophy	7
2.1 Introduction	7
2.2 Three Scenarios	8
2.2.1 Interactive document creation and editing	8
2.2.2 Second Scenario	8
2.2.3 Third Scenario	9
2.2.4 Discussion	9
2.3 Model	10
2.4 Structure and Implementation	11
2.5 Status	13
2.6 Acknowledgements	14
3 Overview of Mint features	15
3.1 Major Omissions	15
3.2 Environments	16
3.2.1 Box Environments	16
3.2.2 Modification of Box Environment Parameters	18
3.2.3 Slug Environments	18
3.3 Tabulations	18
3.4 Labels and Tags	18
3.5 Page Layout	19
3.6 Document Types	19
3.7 Other Features	19
3.7.1 Include and Value	19
3.7.2 Plot and DP	19
3.7.3 Macrogenerator	20
3.7.4 Bibliographies	21
3.7.5 Running Mint	21
3.7.5.1 Cross-Proofing	21
3.7.5.2 Error Reporting	21
3.7.5.3 Performance	21
3.7.6 Examples	22
4 Reference Section	23
4.1 Input conventions	24
4.2 Document syntax	25

4.2.1 Syntactic metalanguage	25
4.2.2 Pseudo-Syntactic properties	26
4.2.3 Nonterminals	26
4.2.4 Production rules common across several document types	29
4.2.4.1 Document environment syntax	30
4.2.4.2 Terminal environment syntax	30
4.2.4.3 Heading environment syntax	30
4.2.4.4 Section environment syntax	31
4.2.4.5 Chapter environment syntax	31
4.2.4.6 Itemization environment syntax	31
4.2.4.7 Title page environment syntax	31
4.2.4.8 Galley environment syntax	31
4.2.4.9 Page environment syntax	32
4.2.5 Document Syntax	32
4.2.6 Altering the syntax	32
4.3 Galleys	33
4.3.1 Defining galleys	34
4.3.2 Procedure Families	34
4.3.3 Font Families	35
4.3.4 Installing a galley	35
4.3.5 Dominating environments	35
4.4 Standard Galley Properties	36
4.4.1 Procedure families	36
4.4.1.1 Terminal procedures	36
4.4.1.2 Heading procedures	37
4.4.1.3 Chapter procedures	37
4.4.1.4 Section procedures	37
4.4.1.5 Itemization procedures	37
4.4.1.6 Title page procedures	37
4.4.1.7 Document type procedures	38
4.4.1.8 Footnote procedures	38
4.4.1.9 Annotation procedures	38
4.4.1.10 Contents procedures	38
4.4.1.11 Miscellaneous procedures	38
4.4.2 Font families	38
4.4.3 Prefixes and postfixes	41
4.4.3.1 Prefixes0Chapters	41
4.4.3.2 Prefixes1Chapters	41
4.4.3.3 Prefixes0Sections	41
4.4.3.4 Prefixes0Items	41
4.4.3.5 Prefixes0TitleEnvs	42
4.4.3.6 Postfixes0Terminals	42
4.4.4 Standard styles	42
4.4.5 The galley parameters for the document types	42
4.4.5.1 Text, form 0	42
4.4.5.2 Text, form 1	43
4.4.5.3 Report, form 0	43
4.4.5.4 Article, form 0	44
4.4.5.5 Thesis, form 0	44
4.4.5.6 Slides, form 0	45

4.4.5.7 Manual, form 0	45
4.4.5.8 Manual, form 1	45
4.5 Box Environment Parameters	46
4.5.1 Standard attributes	46
4.5.2 Additional parameters	48
4.5.2.1 Style parameters for document types	49
4.5.2.2 Additional parameters for title pages	50
4.5.2.3 Itemize and Enumerate	50
4.5.2.4 Maths	51
4.6 Units of length	51
4.6.1 Absolute units	51
4.6.2 Raster lengths	51
4.6.3 Relative lengths	52
4.6.4 Modifying environment parameters	52
4.7 The standard values for the environment values	52
4.8 Slug Environments	60
4.8.1 Face Codes	60
4.8.2 Font Sizes	60
4.8.3 User Face Codes	61
4.8.4 Underlines, Overlines and Eraselines	61
4.8.5 Raster Functions	61
4.8.6 Scripting	62
4.8.7 Overprinting	62
4.9 Fonts	63
4.9.1 Associating Fonts with a galley	63
4.9.2 Modifying fonts	63
4.9.3 Icons	64
4.9.4 New fonts	65
4.9.5 The mathematics fonts	66
4.9.6 Character information	66
4.10 Colours	67
4.10.1 Defining colours	67
4.10.2 Associating colours with objects	68
4.10.2.1 Associating colours with page areas	68
4.10.2.2 Associating colours with boxes	68
4.10.2.3 Associating colours with borders	69
4.10.2.4 Associating colours with characters and lines	69
4.10.3 The order of overlaying	70
4.11 Computations	70
4.11.1 Standard computations	71
4.11.2 Arbitrary computations	71
4.12 Box procedures	72
4.13 Devices	74
4.13.1 Device table information	75
4.14 Presentations	75
4.14.1 The structure of a presentation	75
4.14.2 Defining layout procedures	76
4.14.3 Defining new presentations	76
4.14.4 Making representations	77
4.14.5 Printing a presentation	77

4.15 Standard presentations and printing	78
4.15.1 Standard presentations	78
4.15.2 Printing the standard presentation	78
4.16 Layout procedures	78
4.16.1 Sorting the slugs and boxes	79
4.16.2 Page areas	79
4.16.2.1 Page parameters	79
4.16.2.2 Area parameters	80
4.16.2.3 Values of the page area parameters	80
4.16.3 Actions of the layout procedures	81
4.16.3.1 The Default layout routine	81
4.16.3.2 The TitlePage layout routine	82
4.16.3.3 The Contents layout routine	82
4.17 Macrogenerator	83
4.17.1 Input conventions	83
4.17.2 Defining macros	84
4.17.3 Access to system values	85
4.17.4 Deferred Macros	85
4.18 Standard Macrogenerator Facilities	87
4.18.1 Predefined Macros	87
4.18.1.1 Special macros	87
4.18.1.2 Bibliographic macros	88
4.18.1.3 Counter macros	88
4.18.1.4 Galley macros	88
4.18.1.5 Presentation macros	89
4.18.1.6 Syntax macros	89
4.18.1.7 Computation macros	89
4.18.1.8 Index macros	89
4.18.1.9 Maths macros	90
4.18.1.10 Border and colour macros	90
4.18.1.11 Extra macros	90
4.18.2 System attributes accessed via @Value	90
4.19 Bibliographies	91
4.19.1 Citation collections	91
4.19.2 Citations	92
4.19.3 Causing the bibliography to appear	92
4.20 Indexes	93
4.20.1 Index collections	93
4.20.2 Index entries	93
4.20.3 Causing the index to appear	94
4.20.3.1 The Style 1 indexing routine	95
4.21 Prefixes and postfixes	96
4.21.1 Standard prefixes	96
4.22 Counters and Labels	98
4.22.1 Overview of Counters	98
4.22.2 Counter manipulations	99
4.22.3 Labels	100
4.22.4 Referring to labels	100
4.22.5 Conversions	101
4.22.6 Undefined labels	102

4.23 Standard Conversions and Counters	102
4.23.1 Conversions	102
4.23.2 Pseudo-counters	103
4.23.3 Non-basic conversions	103
4.23.4 Counters	104
4.23.4.1 Counters common to all document types	104
4.23.4.2 Counters in document types with footnotes and annotations	104
4.23.4.3 Counters in document types that have chapters	104
4.23.4.4 Counters in document types that have sections	104
4.24 Miscellaneous layout statements	105
4.24.1 Spacing statements	105
4.24.2 Page commands	105
4.24.2.1 Page headings and footings	105
4.24.2.2 Page offsets	106
4.24.2.3 Page skips	106
4.24.3 Tabulations	106
4.24.4 The <code>Align</code> environment	107
4.24.5 The <code>Describe</code> environment	108
4.25 Cross proofing	109
4.26 Borders and Border Styles	110
4.26.1 Border Styles	111
4.26.1.1 Lines	111
4.26.1.2 Patterns	111
4.26.1.3 Border Styles	111
4.27 Mathematical Typesetting	112
4.27.1 Basic Concepts	112
4.27.2 Simple formulae	114
4.27.3 More complex formulae	115
4.27.3.1 Formula types	115
4.27.3.2 Labelled equations	117
4.27.4 Advanced concepts	118
4.27.4.1 Mathematical fonts	118
4.27.4.2 Changing fonts	118
4.27.4.3 Defining symbols	119
4.27.4.4 Inflected symbols	120
4.27.4.5 Grouping subformulae	121
4.27.4.6 Controlling the style	121
4.27.4.7 Mathematical environment parameters	121
4.27.4.8 Tabular layout of formulae	123
4.27.4.9 Equation counters	125
4.27.5 Really advanced features	126
4.27.5.1 Mathematical layout vectors	126
4.27.5.2 Styles	127
4.27.5.3 Types	128
4.27.5.4 Spacings, etc.	128
4.27.5.5 Mathematical font parameters	129
4.28 DP and Plot	130
4.29 Errors	131
4.30 Quirks and Oddities	131



Part One

Getting Started

This document describes version 2A(21) of Mint, a document preparation system that has been written as part of the Spice project. Mint has been written as a research vehicle for exploring document preparation, and interactive document preparation in particular. However, I feel that the current version of Mint, although it does not have interactive features, is nonetheless a usable tool, and therefore that it is suitable to release it for evaluation and use by a wider community. In making this release, I am making a commitment to providing a stable and maintained system <<+ disclaimers about improvements>>

The document is organized as follows. The introduction provides an overview of the system and gives operating information; the information provided here should be sufficient to allow the casual user to prepare documents on the Perq of the same quality as those produced by Scribe. The next section is a copy of the first part of a paper that was prepared for the November 1982 Spice Industrial Affiliates meeting [Spice document S148]. It reviews the goals of the document preparation research project, and is included here to provide a context for understanding the nature of Mint, and to indicate the likely direction of future developments. The third section contains a review of the design and features of Mint: this section should be read by anyone wishing to make extensive use of the system. Finally, the last section contains reference material.

1.1 Mint and Scribe

At a superficial level, Mint resembles Scribe — it takes as input a .Mss file, and produces a formatted document for some device. Most .Mss files that just use the basic Scribe commands — those in sections 1 to 6 of the Scribe manual — will be accepted by Mint, and will produce results that resemble very closely those from Scribe. In addition, there are Mint equivalents for most of the rest of the Scribe facilities, though these are obtained in different ways.

The simplest way of using Mint then is to treat it as though it is Scribe, and, if it doesn't produce the expected output, to read the more detailed description in section 3, or look up the description of the feature in section 4. In this way you will discover the subset of the facilities that are common to both Scribe and Mint. However, you won't get the best out of Mint if you simply treat it as Scribe, so you ought to read section 3 anyway.

1.2 Getting Started

I assume that you already have an account on a Perq, and are sufficiently familiar with the machine to boot POS on it, run programs, create and edit files, etc. If not, get in touch with David Nason, x2585. He will give you an account and introductory documentation.

1.2.1 Loading Mint onto a Perq

Normally Mint will be on the Perq, and there will be no need to reload it. If, however, Mint has not yet been installed, or the version currently available on the Vax is a more recent version than that on the Perq, then you should load it as follows:

```
@update <RETURN>
Special options?: [n] <RETURN>
Remote directory: /usr/spice/pos/mint/seg <RETURN>
```

(when complete, Update will display the Mint change log). If the Update command file `update.cmd` cannot be found the Perq will return an error message:

```
** Command file not found: Update
```

If that happens, you must retrieve a new version of Update as follows:

```
cmuftp r /usr/spice/pos/gSUPD10 gsUPDATE
cmuftp @gsUPDATE
```

and then start again. (If your Perq is on the 3 Mbit EtherNet, replace `gSUPD10` by `gSUPD3`)

1.2.2 Running Mint

To invoke Mint, type

```
mint
```

If the Perq responds with

```
** Loader-F-Mint.RUN not found
```

then Mint has not been loaded, and you should start again at section 1.2.1. If Mint is loaded, two windows will appear on the screen. The upper one is used by Mint when it outputs a formatted document to the screen, and also when it is being run in debug mode (normally this will not be the case); the lower window contains the dialogue and error messages.

Mint asks for the name of the file to be formatted. If you have just been editing a file, Mint will assume that this file is to be used as the default. The file name can be typed in as a full path name or just as the file itself, in which case Mint will search for the file on the current search list. The extension `.Mss` may be omitted.

After it has asked for the file, Mint will ask for which device the output is intended. Currently there are two devices available — the Perq itself, and the Dover. Mint then asks a couple of questions about debug output — respond with carriage return to both questions. Mint then goes about its business. As it formats the document it places the current file and line number in the banner of the lower window, but as it only does this on each change of environment, there may be periods of several seconds when the screen does not change. If the output device has been specified to be the Perq, the document galleys are written into the top window. Section numbers will appear as question marks, and several galleys may get overlaid. Do not worry about this — what you are seeing is only an intermediate version of the document. Error messages appear in the lower window and are written off to an error file. More details are given in the next section.

When the document has been completely formatted, Mint will ask the question

```
Printing Device: Perq, Dover, Report, Quit:
```

You now have the option of choosing the device on which to view the document. If you specify the Dover, then a press file will be created, with extension `.Press`, which can be shipped to the Dover for printing. If you specify the Perq, the message

```
Which page (<integer>, All, Dover, Quit, <CR> for next):
```

will appear. Typing in a page number will cause that page to be displayed on the screen, typing upper or lower case `a` will cause all the pages to be displayed, and typing a carriage return will cause the next page to be displayed. You may exit by typing `Q` or `q`, when the question

```
Printing Device: Perq, Dover, Report, Quit:
```

will reappear. If you type `R` or `r` then a bug report will be prepared for the maintainer (i.e., me); more details are given in section 1.3.

Note that the viewing device may differ from the target device given in the first request. This allows you to “cross-proof” a document intended for one device on another. Normally you would want to view Dover output on the Perq screen in order to save the delay in shipping a Press file to the Dover; if it is satisfactory you can create the Press file without reformatting the document. If you are cross-proofing on the Perq, you can also select which page you want to incorporate into the Press file by typing the characters `D` or `d` when Mint asks for which page to print.

In cross-proofing mode, each character appears on the viewing device in the position that it will have on the target device, and diagrams are scaled appropriately. However, the fonts used on the two devices will not be identical, so that the output will not be an exact representation. See section 4.25 to see how to map different device fonts to improve the appearance of cross-proofed documents.

1.3 What to do in case of trouble

Error messages generated by Mint fall into four classes.

Warning	Mint issues a warning if it finds something suspicious in the input, but which is quite legal. The output from Mint may be satisfactory, but you should investigate the reason for the warning, as it may indicate some misunderstanding.
Error	Errors occur when Mint is not able to process the text as you ask, but it is able to take some corrective action and continue. It is unlikely that the output will be what is desired, though.
Heresy	A heresy indicates that there is some serious problem that Mint is not able to fix in a reasonable way. In general these are caused by internal problems, and usually indicate Mint bugs, though improper use of the advanced facilities described in section 4 also can cause them. As in all organizations, one can continue after a heresy, though subsequent actions by the system cannot be predicted.
Fatal Error	These occur when Mint has discovered an internal error that will cause it to fail if it continues. Usually these are caused by overflow of internal tables, and can be fixed fairly easily in future releases.

In all cases, the error message is written in the lower window, and sent off to a file with the extension `.Error`, with an indication of the location of the error. In the case of a warning or an error, Mint continues; in the case of a heresy or fatal error it halts with the message

```
Quit, Continue, Report or Alter Flags (Q, C, R, A) [R]:
```

Unless you are a Mint maintainer (and you are not) you should type `Q` or `q` (or `R` or `r`, which under these circumstances have the same effect).

After a fatal error or a heresy, copies of the `.Mss` file and the `.Error` file are taken, and you are invited to report the problem to the maintainer. Please do this unless you don't want me to see the `.Mss` file. For more details, see the next section.

You can halt Mint at any time by typing `↑C`. The message

```
Quit, Continue, Report or Alter Flags (Q, C, R, A) [C]:
```

appears. Quit, continue or report, as you wish. (You can alter the Debug Flags also, but then you are out on your own). Note that `↑S` and `↑Q` do not work for output into the upper window.

1.3.1 Reporting Errors

Even though it is written with all the usual attention to detail, and with the needs of the consumer always in mind, Mint does have bugs, as well as several exotic and unusual features. It will only be possible to tame

the beast if unusual occurrences and bugs are reported. Please (please, please) adopt the following procedure.

If Mint exhibits unusual behaviour, exit from Mint by typing R or r to the message

Printing Device: Perq, Dover, Report, Quit:

or, if you want to halt Mint, type ↑C and then R or r to the message

Quit, Continue, Report or Alter Flags (Q, C, R, A) [C]:

Mint will enter a phase that asks for various pieces of information to help me find the error. To do this a copy of the .Mss file has to be (automatically) taken; if you do not want me to see the .Mss file, exit from this phase by typing Control-Shift-C. Horrible things will happen, but your confidential .Mss file will be secure. A somewhat more graceful way of exiting is to type Control-Shift-D to enter the debugger, and then to quit from the debugger.

1.4 The Future

There is one. It still exists.



Part Two

Design Philosophy

2.1 Introduction

In our present time-shared systems, document preparation occupies a significant position. The tools that are used are editors of various kinds, document formatters such as Scribe and T_EX, and several related tools such as drawing packages. There is every reason to believe that the use of computers for preparing documents will increase, and that document preparation tools will play a significant role in personal computing environments. However, I believe that we need to make significant improvements in the tools before we can take full advantage of personal machines, both by making use of our increasing understanding of how to do document production, and by integrating the tools we have.

I believe that now is the right time to start looking at these issues. We need to be able to integrate text and graphics in a coherent fashion, allowing editing of both text and drawings from within the same editor. We need to be able to provide better tools for specialized typesetting needs, such as mathematical typesetting, advertising typesetting, tabulations, etc. We need to allow multiple authors to work on a document, and provide them with the tools necessary to review changes and annotations. We need to be able to make better use of high quality printing devices. Finally, we need to reconsider what is the most appropriate way of presenting information, and the relations that exist between those pieces of information. This presentation should also provide an interactive environment that allows rapid redisplay, formatted and tailored to the needs of the user.

In this paper I will try to address some of these issues from the standpoint of the Spice Project. I will do this with the attitude of mind in which practicality is important — that the ideas expressed must be implementable in the software technology framework we have now, and that the tools that are produced must be attractive and usable by the Spice community.

The paper is laid out as follows. Section 2.2 gives three scenarios in which interactive document preparation tools are used, in order to illustrate some of the facilities I regard as important. Section 2.3 then describes a model for document production by introducing a set of abstractions and operations on these abstractions that allow good quality documents to be created. In section 2.4 I describe the implementation structure of the system, both as it now exists and as it may develop. Finally, in section 2.5 I describe the current status and plans for the future.

2.2 Three Scenarios

These scenarios cover increasingly complex situations in document preparation, and illustrate the range of facilities that are required.

2.2.1 Interactive document creation and editing

The user is sitting at a personal computer, creating and editing a piece of text. As the text is typed in, the interactive document preparation tools format it — they fill out the lines, adjust the margins, add additional information such as section numbers, and switch automatically to the appropriate type face for headings, programs, etc. While typing the text, the author can call on a number of other tools — spelling checkers and bibliography tools, for example. At any time he can display the document's structure, possibly presented as a table of contents, or as a tree structure. He can, if he wishes, create the document structure first, and then use the document preparation tool to fill in the section stubs. The editing that he can perform is not only on a character and line basis. He can also use the tool's understanding of the structure to edit whole sentences, paragraphs or sections.

When the document is complete, the user may store it, together with the internal formatting information, send it to a colleague, or print it. At no time is the internal formatting information seen.

2.2.2 Second Scenario

The author now wants to create a more complex document than the text document created in the first scenario. He wants to incorporate tables and mathematical formulae, graphs and line drawings, and he wants to have a more elaborate page layout than the one provided by default. Some of the graphs and line drawings exist in preprocessed form in libraries, and these he needs to be able to incorporate into the document; others he wants to create and edit along with creating the text. The document preparation tools must allow any part of the document to be edited, by providing the appropriate editing primitives for each type of information — line editing primitives for the graphs and line drawings, text editing primitives for the text. Some sorts of information have properties of both text and line drawings: mathematical formulae and tables, for example, consist of predefined symbols that can be laid out using simple rules of formatting, but the result is usually not entirely satisfactory, and the positions of the symbols need to be adjusted, or alternative layouts used, depending on taste and requirements. To do this the author uses editing primitives similar to those for graphical editing. However, there is a compensation: having finely tuned some particular formula, the system now uses that same finely-tuned layout for all similar formulae.

A similar situation arises with page layout. A default set of rules may be able to lay out a page in a satisfactory fashion for non-archival documents, but be incapable of producing output satisfactory for books, for example. In this latter case, the user is prepared to spend effort in organizing the pages. The document preparation tools allow him to move diagrams on the page, causing text to flow round tables, expand or contract the spacings between words and lines to avoid pages with too much blank space at the end of

paragraphs, or design a page layout in the way a graphical artist or advertising copywriter might. In all cases, the author specifies the relevant features of the appearance, and then relies on the document preparation tools to reformat the rest of the document to meet those constraints. For example, it may move information from page to page, or choose similar page layouts for all the other corresponding pages in the document.

2.2.3 Third Scenario

In this scenario, a paper is being written by several co-authors. Each of them plays the roles of author by adding original text, and of editor by adding annotations and suggesting amendments for review by others¹. The document preparation system now has to function as a medium of exchange of information between the authors, allowing suggestions to be pencilled in, proposed rewordings to be made, and significant modifications of the structure to be suggested. The author makes his annotations in several different ways — sometimes by using a style similar to text insertion in a text editor, sometimes using a pointing device to simulate proof readers' marks. At any time the author can examine the document as it would appear with his suggested rewordings, or go back to some previous version and see that. The annotations are classified by author, so he can also see the annotations of any particular author. If the author agrees with a suggested change, then he accepts it, and the 'delayed.edits' implicit in the annotation get made.

2.2.4 Discussion

The first scenario assumes a collection of tools that already exists in several systems, so proposes nothing novel. However, I would like to draw an analogy between this scenario and a similar one that would involve interactive programming. In both cases the users of such systems are concerned with high-level specifications of the behaviour of the systems, rather than the details of how that behaviour is to be obtained. In both cases the tools understand the structure of the input — the document and the program — and use their understanding to determine how far changes to the high-level representation need to be propagated through the low-level representation. Finally, in both cases, high-level specification languages are used to reduce the amount of detail the user requires to use the tools: in the case of the interactive programming environment the high-level specification language is the programming language; in the case of the document preparation system it is the implicit language given in the formatting commands.

The second scenario shows that while high level specifications of document appearance may satisfy the majority of requirements, there still needs to be a facility for fine-grained control. However, it is important that this fine-grained control is not given by providing direct access to the lowest-level formatting primitives used by the system (which is equivalent to providing a high-level language and compiler for writing the programs, and DDT for doing the optimization), but that there are several distinct levels of abstraction, each providing a suitable collection of abstract objects and operations.

¹ Although the task described here involves several authors, even a single author has several perspectives on a paper he has written, and will go through a style of interaction with the document similar to that described here.

The third scenario shows that interactive document preparation potentially needs the same range of tools that multi-programmer projects require. These tools however must be more supportive of the 'drunkard's walk' technique of converging on a document satisfactory to its authors and audience. The basis of this is that all amendments and annotations should be saved with the document, which must form the medium of interchange for the evolving ideas. In addition any set of annotations should be able to be reviewed using a number of different styles of physical presentation, and finally constraints on the relationships between items of information in the document should be checked as a part of the document preparation action when changes are made². Finally, this scenario indicates that a document should properly be considered as a collection of items of information, together with a record of the relations between these pieces of information, and that there may be several different presentations of the document, rather than a single snapshot, each tuned to the reader's needs and the capabilities of the output device.

2.3 Model

Having presented above a view of the document preparation task that was unconstrained by issues of implementability, I now wish to reiterate that my purpose is to build a collection of document preparation tools that have a practical value within the Spice project. Since the field of computer-aided document preparation is relatively new, well understood models and techniques are not available. I have therefore drawn on my experience in other areas for generating a model. It is quite clear to me that the particular model I have chosen will need a lot of refinement to be able to achieve the goals implied above; however, I believe that we need the experience now, and that the time is ripe to experiment with interactive document preparation tools.

The model below has been derived from two areas. My (very imperfect and incomplete) knowledge of traditional typesetting technology has led me to a set of abstractions that I believe capture the concerns for efficient representation of documents without sacrificing ease and flexibility of editing. My experience as a language designer and implementer has given me an appreciation of the power of the abstraction mechanism, and an understanding of the importance of choosing the right information to bind and the right time to bind it.

There are three basic abstractions that are used: the galley, the box and the slug. In the printing trade, a slug is a collection of pieces of type that form a single printed line. Slugs are bound together into boxes that represent a collection of lines in a single textual unit — a paragraph, heading, caption, etc. Gaps are inserted between the slugs to adjust the line spacing, and at the left and right hand sides to adjust the margins. Finally, the boxes are laid out into galleys — long trays the width of a printed page, which represent the appearance of the document before it is broken into pages. Typically there are several different kinds of galley. For example there will be different galleys for footnotes and for block diagrams.

² Thus trivially, if a section is deleted, then the user should be informed that there are references in the rest of the document to the deleted section. Less trivially, there are implicit relations between parts of the document that are more subtle than such explicit cross-references, and a document preparation system should keep a record of them as a part of the structure of the document.

It is from the galleys that galley proof copies are obtained. The cost of making editorial changes to the galley is generally the cost of making changes to a single slug, or at worst, several slugs in a single box. The final stage of typesetting is that of page layout: the boxes in the galleys are laid out into page boxes by collecting the slugs and boxes from several galleys (such as the footnotes galley and the line drawing galley), if necessary splitting boxes apart to place some of the slugs on one page and the rest on the next page. Headings and footings are also added. Editorial corrections at the page proof stage are potentially much more expensive than at the galley proof stage since they can cause several pages to be remade. Printers frequently will employ extraordinary strategies to allow an editorial change to be made without disturbing the page layout (even to the extent of rewriting sentences to alter their length if that allows the changes to be localized on one page).

Partly because traditional printing technology has evolved to meet a similar set of concerns present in computer document production, the techniques form a basis for the abstractions that are suitable for a computer based system. The traditional techniques provide a range of representations of information, with a similar range of costs for modifying the representations. Within each representation (the slug, the box, the galley and the page) a different piece of knowledge is encapsulated and a different set of techniques needs to be applied to alter and manipulate these representations. The galleys contain information that has implicit relations with information in other galleys; when page layout is performed these relations are made explicit, either by the relative positions of the boxes, or by explicit footnotes, for example.

The model I have adopted is closely based on this one. The system is divided into two parts: the low-level part and the high-level part. The low-level part comprises facilities for interpreting the input and creating slugs from it, and for displaying the slugs on request. Since a document comprises several sorts of information, each of which needs different interpretation — text, line drawings, halftone pictures, etc. — there are several independent low-level interpreters. One of the responsibilities of the front end of the formatter is to determine which interpreter has to be fed the input; thereafter the slug interpreter operates independently. The high-level part of the system is responsible for manipulating the boxes and slugs, and does not need to examine their content. This part needs information about the size of the boxes, whether a particular box can be split into individual slugs for performing page layout, and whether a box can be floated in the document, for example. Finally, pages are created from the boxes in the galleys by several page layout modules.

2.4 Structure and Implementation

Because the goal of producing good quality interactive document production tools is open-ended, I established a major subgoal — that of building a working non-interactive document preparation system of capability comparable to Scribe, but with the appropriate internal structure to allow experimentation and incorporation of other document preparation tools. Since feed-back from users will be important, I have chosen as an input language to the system the same language as is used by Scribe. However, even though the system operates like Scribe in terms of its input and output, its internal structure is very different.

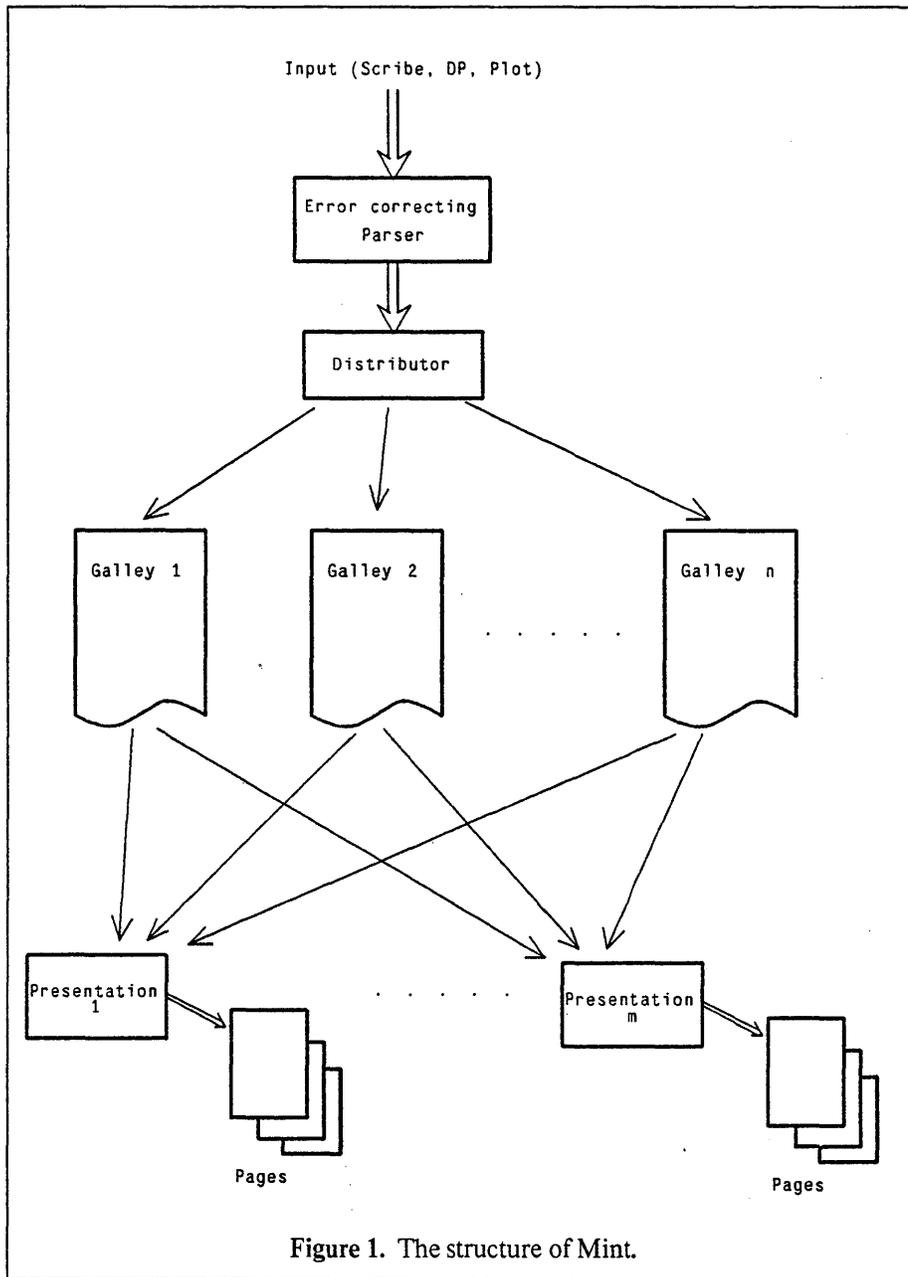


Figure 1. The structure of Mint.

The principal components of the system are shown in the figure. Since the Scribe language is somewhat context-sensitive, and only loosely defined, it is first converted into a context-free language, rich enough to express the structure of the document, by means of a parser that uses traditional techniques of syntactic error-correction to convert the input into an appropriate representation. The techniques that are used are sufficiently general purpose that arbitrary context-free document structures can be defined (the parser is table driven), and other front-ends could be substituted to allow other input languages.

Depending on the node type of each of the nodes, the node contents are sent to the appropriate galley, where the contents are formatted according to global style parameters associated with the galley. The node also carries information about which interpreter is to be used to create and display the slugs that are created. The interpreters are responsible for creating the slugs, placing them into the boxes, and displaying them when requested.

After the galleys have been created, page layout is performed. The controlling abstraction is the *presentation*: a collection of rules and procedural knowledge which specify which boxes and slugs from which galleys are to be collected together into the pages, how the pages are to be laid out, and how the relationships between the information in the pages is to be displayed. Each presentation is independent, and several can exist simultaneously. The pages of a presentation can be converted into Press files, displayed on the Perq screen, or turned into compact representations which have had the structural information eliminated, for use as non-editable messages.

2.5 Status

The first usable version of the non-interactive version of the formatter is now operating. In designing and implementing it I have made a balance between the usefulness of the features I have been adding, and their use in illustrating and testing design decisions. Thus while the system is not complete, it is now able to replace Scribe competitively for about 90% of Scribe use. With the exceptions noted in the appendix, bringing the non-interactive version up to Scribe class is reasonably mechanical.

There are two particular areas, however, where the value of the internal structure of the formatter has been apparent. First, new environments can be defined by programming at an appropriate level of abstraction: that of the manipulations on the slugs and the boxes. As an example, I wished to have an environment that would allow me to place a commentary on a piece of text at the side of the text, using a different, smaller font. Although the slugs in the text on which I am commenting are independent of the slugs of the commentary, the boxes are not, and need to be placed side by side. An example of such a commentary is on page 16. The core of the system that produces this output is the routine that gets called by the galley to create the box. The routine is listed below.

```
procedure BoxCommentary0 (BP: IBoxPtr; M: IModListPtr);
  var
    E: IEnvPtr;
    B: IBoxPtr;
begin
  AllocBoxPtr (B);
  AllocEnvPtr (E);
  CreateEnvRecord (Commentary, E, M, BP);
  StartLoop;
  RepeatUntil (BoxEnded);
  GetNewBox (B, BP, E);
  ParseBox (Commentary, B);
  IfTrue (HaveEnv (Gloss));
    ParseBox (Commentary, B);
  EndIf;
```

```
        SetBoxYSize (B);  
    EndLoop  
end;
```

While I do not expect casual users of the system to program it, it will be necessary to have document designers who can maintain and improve the library of layouts that are available. The level of abstraction implied by this routine is a suitable one.

The second example of the capabilities of the system came during a couple of experiments that were performed recently. In order to test my assertion that independent interpreters could be incorporated easily into the system, without need for any more than the most casual programming on each side of the interface, Ivor Durham and I integrated Plot into the formatter, and Dario Giuse and I integrated DP. The total amount of time needed was little more than a man week, and now both Plot diagrams and DP drawings can be brought totally under the control of the formatter's high-level manipulations. Examples are given in later sections.

2.6 Acknowledgements

Many people have contributed to the development of Mint. Special mention must be made of Ivor Durham, Dario Giuse and John Renner, who have contributed ideas and comments, and have assisted in the implementation; and of Rob Witty of the Science and Engineering Research Council, for generously providing me with a working environment in England during which many of the ideas of Mint were developed. In addition, everyone working on the Spice project has contributed, either directly or indirectly, by providing the stimulus necessary to drive this effort. Finally, I owe a debt of gratitude to Brian Reid, the author and implementer of Scribe, who steadfastly maintained that document preparation was a legitimate scientific pursuit, and who paved the way for many of the ideas I have incorporated into Mint.

Part Three

Overview of Mint features

I have used two guiding principles in designing and implementing Mint.

- Although I have adopted the notions of environments and environment parameters from Scribe, and I have attempted to keep close to the same semantics that Scribe has, the internal structure of Mint is very different from that of Scribe. This means that there are many places where similar effects are achieved differently. In some cases I have not felt it worth my while (at this stage) disguising these differences from the user, though I am sure they could be hidden.
- I have chosen to implement a subset of the features that will eventually be needed, in order to spread my effort across all the design issues. Extending the capabilities of Mint to include these omitted features should be a fairly mechanical task.

The description below is not intended to be a User's Guide, though users with a knowledge of Scribe and some patience should be able to make use of the system from this description.

3.1 Major Omissions

The following facilities have not been implemented in Mint.

Indexes	I do not see any difficulty in providing these. The index for the reference section was produced semi-automatically, so I am confident that the mechanisms work satisfactorily; all that is now required is to make the mechanisms available to users.
Exotic devices	Mint generates output suitable for the Perq, and Press files for the Dover. Extending Mint to generate Press files for any other device that takes Press files is trivial, and I will do that shortly. I also have a grubby interface to the Canon printers that you don't want to know about.
Definitions	To make life easier for myself, I programmed the calls to the style-setting routines into the code, rather than use a definitions file as Scribe does. I have extended Mint's prescanner to accept definitions in documents, and it is now possible to alter most of the defaults that I selected. As I get more familiarity with users' requirements, I will write a set of definitions files to modify Mint's actions. However, this isn't the document to describe these facilities; read the reference manual to find the information.
Tables of Contents	I do not see any difficulty in providing these.

3.2 Environments

In Mint environments are classified into two sorts — those that control the general appearance of boxes, and those that control the appearance of slugs. They will be termed *box environments* and *slug environments* respectively. (As it turns out, slug environments correspond to the Scribe environments that have `TabExport` set to a default value of `True`.)

3.2.1 Box Environments

The following are implemented: `Display`, `Centre`³, `Example`, `FlushLeft`, `FlushRight`, `Verbatim`, `Format`, `Quotation`, `Itemize`, `Enumerate`, `Description`, `MajorHeading`, `Heading`, `SubHeading`, `Section`, `SubSection`, `Paragraph`, `Figure`, `Caption`, `Report`, `Thesis`, `Article`, `Manual`, `Slides`, `Foot`, `TitlePage`, `TitleBox`, `ResearchCredit`, `Abstract`. In addition, environments particular to Mint are `DP` and `Plot`, which are described in section 3.7.2, and `Commentary` and `Gloss`, which allow text to have a gloss against it, as the following example shows:

For the elaboration of a discriminant constraint, the expressions given in the discriminant associations are elaborated in some order that is not defined by the language; the expression of a named association is evaluated once for each named discriminant.	This rule and the rules defining the elaboration of an object declaration ensure that the discriminants always have a value. In particular, if a discriminant constraint is imposed on an object declaration, each discriminant is initialized with the value specified by the constraint.
---	--

A complete list is given on page 26. Box environments take a number of parameters that are set up with the defaults you would expect from Scribe. The environment parameters are as follows.

Parameters similar to those of Scribe

`Width`, `Above`, `Below`, `Need`.

Parameters that differ slightly from those of Scribe

Line spacing	<code>Gap</code> specifies the distance between the bottom of one line and the top of the next. Used instead of <code>Spacing</code> .
Justification	Justification in Mint is done with four parameters that take values from (<code>True</code> , <code>False</code>). The parameters are <code>JustifyLeft</code> ; <code>JustifyRight</code> which control the justification of all except the last slug of a box, and <code>JustifyLeftLast</code> , <code>JustifyRightLast</code> which control the justification of the last slug.

³ I use this spelling to revenge myself for all those times in the past that I have had to re-Scribe documents for misspellings. If you don't want to use it, you can write `@define (center = centre)`

- Margins** These are controlled by `LeftMargin` and `RightMargin`. The indentation of the first line is controlled by the *document style* parameter `indent` that can only be set in the `@make` command.
- Underlining** It is possible to simultaneously and separately underline, overline and draw delete lines through text, spaces and punctuation. The parameter `Underline` takes values from the set (`None`, `NonBlank`, `All`, `AlphaNumeric`, `OverNonBlank`, `OverAll`, `OverAlphaNumeric`, `EraseNonBlank`, `EraseAll`, `EraseAlphaNumeric`). Several non-conflicting values may be set simultaneously.
- Page layout** Page layout is controlled by the parameter `PageStyle`. It takes values from (`Skip`, `Default`, `TitlePage`). You probably shouldn't fiddle with this yet.

Parameters that differ significantly from those of Scribe

- Fonts** With each galley is associated a two dimensional array of fonts, indexed by `FontSize`, taking values (`11`, `1`, `n`, `s`, `ss`), for extra large down to extra small, and by `FaceCode`, taking values from (`r`, `i`, `b`, `c`, `g`, `t`, `p`, `z`, `a`, `f0`, `f1`, `f2`, `f3`, `f4`, `f5`, `f6`, `f7`, `f8`, `f9`). Fonts are associated with (a subset of) the elements of the array, and are selected by independent use of `FontSize` and `FaceCode`.
- Tabulations** Tabulations are a part of the box environment. `TabSet` takes a distance as parameter, and sets (another) tab at that point, `TabDivide` divides up the width of the box into some number of tabs, and `TabClear` clears all the tabs. Defaults are inherited from the parent environment in the usual way. Tabs can also be set dynamically, see page 18.
- Borders** Mint allows you to draw borders round a box. Two parameters are used: `Border` sets the width of the frame around the box, and `BorderStyle` specifies the pattern drawn in the frame. (You can draw borders which are wider than the frame, but then the border may overprint some of the box contents.) Border styles can be defined to satisfy the most exotic of tastes, but the details are beyond this document. However, one border style that is freely available is `Width1`, which draws a boring narrow line around a box.
- Rasters** If the device you have can paint using one of several raster functions, `RasterFunction` specifies which it will be. Since the functions are device-dependent, I have specified the ones for the Perq, Dover and Canon printer only. For the Perq and Canon, use (`RRp1`, `RNot`, `RAnd`, `RAndNot`, `ROr`, `ROrNot`, `RXOr`, `RXNor`), for the Dover use only `ROr` (the default anyway).
- Colours** There are two parameters: `ImageColour` and `BackGroundColour`. These are not a great deal of use at the moment, since both the Perq and Dover are not very good at colours. Currently they take values from the set (`White`, `Grey`), and allow backgrounds to be filled in.

3.2.2 Modification of Box Environment Parameters

The box parameters can be altered locally, or can be altered globally using `Define` and `Modify`. Mint nests modifications and definitions. Length parameters can be set to absolute lengths, using `in`, `ins`, `inch`, `inches`, `cm`, `cms`, `point`, `points`, `mica`, `micas`. They can also be set relative to existing values using `+`, `-`, `*`, `/`, or using `Raster`, `Rasters` which are device-specific, or `Em`, `Ems` and `Line`, `Lines` which are font-specific. During page layout, `PageHeight` and `PageWidth` can be used.

3.2.3 Slug Environments

These are generally identical to those of `Scribe`, but they cannot be modified. The environments are:

Face codes	<code>r</code> , <code>i</code> , <code>b</code> , <code>c</code> , <code>g</code> , <code>t</code> , <code>p</code> , <code>z</code> , <code>a</code> , <code>f0</code> ... <code>f9</code> .
Font sizes	<code>ll</code> , <code>l</code> , <code>n</code> , <code>s</code> , <code>ss</code> .
SomeScripts	<code>+</code> , <code>-</code> .
Raster functions	<code>RRp1</code> , <code>RNot</code> , <code>RAnd</code> , <code>RAndNot</code> , <code>R0r</code> , <code>R0rNot</code> , <code>RX0r</code> , <code>RXNor</code> .
Overprinting	<code>Ovp</code> .
Underlining	<code>u</code> , <code>ux</code> , <code>un</code> , <code>o</code> , <code>ox</code> , <code>on</code> , <code>e</code> , <code>ex</code> , <code>en</code> .

3.3 Tabulations

Mint supports both `@\` and `@↑`. The other tabulation facilities of `Scribe` are incorporated into more general facilities for laying out tables. (See, for example, the tables in section 4.4.2.)

3.4 Labels and Tags

Mint currently treats both of these in the same way. Unlike `Scribe`, a label simply defines some position in the document. This position can be displayed in a number of ways, depending on how the label is referred to. In general, a *counter* needs to be specified; these are similar to those of `Scribe`, but a detailed description is outside the scope of this document. The casual user can assume the following are defined:

<code>@ref(Label)</code>	Yields the section/subsection number, as <code>Scribe</code> .
<code>@pageno(Label)</code>	Yields the page number of the label.

It is possible to change the conversion style of counters (e.g. to Roman or alphanumeric); see the reference manual for details.

3.5 Page Layout

Currently there are two page layouts available in the standard presentation; one for title pages, organized similar to that of Scribe; and the default layout, which places footnotes at the bottom of the page, numbered from 1. Headings and footings are included by using the box environments `PageHeading` and `PageFooting`. The slugs in these boxes are placed consecutively at the head and foot of the page, until the slugs run out, when the box is restarted. You get alternating headings or footings on opposite pages by having two boxes in the environment; alternatively the cycle length can be longer. See, for example, the footing that starts on page 16.

3.6 Document Types

Mint supports the following document types.

Text	This is the default document type. It provides a simple, unsectioned document type, with wide spacing between lines.
Report	This document type has numbered sections.
Article	Similar to <code>Report</code> .
Thesis	This document type has numbered chapters as well as numbered sections.
Slides	This lays out slides using fonts that are suitable for view-graphs.
Manual	Similar to <code>Thesis</code> . This document is a manual.

3.7 Other Features

3.7.1 Include and Value

`Include` takes a file name or full path name; `Value` takes one of (`Date`, `Time`, `TimeStamp`, `Version`, `SourceFile`, `Device`, `DocumentType`).

3.7.2 Plot and DP

The output of both `Plot` and `DP` can be incorporated directly into a document by using the appropriate environment, without the need to go through a `PressEdit` cycle. To include a `DP` drawing, use:

```
@begin(Figure, Width = 3inches)
@DP[@include(Mouse.DP)]
@Caption[Example of a DP drawing]
@end(Figure)
```

which produces the following result.

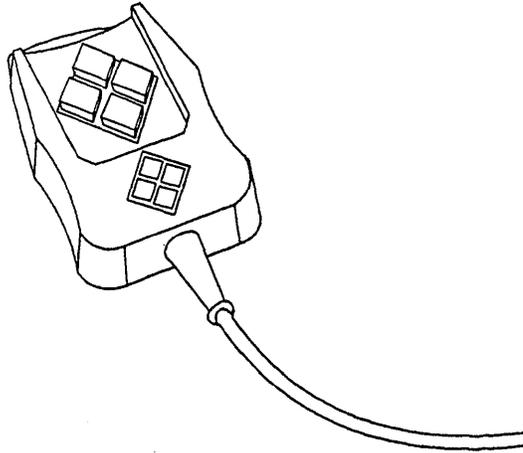


Figure 2. Example of a DP drawing

Similarly, to incorporate a plot, run Plot with its output formatted for the device “generic” and use:

```
@begin(Figure, Width = 3inches)
@Plot[@include(MyPlot.Plot)]
@Caption[Example of a Plot]
@end(Figure)
```

The results will look best if the raster size specified for the generic device is the same as that for the target output device, and if the size specified is the same as that required. However, Mint will scale the diagram if necessary. It is possible to use the Plot and DP environment in all the usual places⁴.

3.7.3 Macrogenerator

Mint supports a general-purpose macrogenerator, which allows all the usual facilities. It is used mainly as a front-end to Mint, to help implement several commands; it *shouldn't* be used like the Scribe macrogenerator, since many of the features that that macrogenerator provides are built into Mint.

⁴ For example, it is quite easy to include a diagram in a footnote, as below, thereby relieving the tedium of what are usually unnecessary annotations.

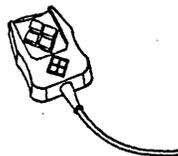


Figure 3. A very small mouse

At the resulting exotic production

3.7.4 Bibliographies

Mint supports a general bibliography feature, which is a specialized form of a *note* facility. However, this document is not the place to describe it. Refer to the reference manual for details.

3.7.5 Running Mint

The user interface has changed considerably since the first version of this document was prepared, and it is not possible to describe all the facilities in detail. The simple mechanics of running Mint can be found in section 1; other features are described later. However, the following should be noted.

3.7.5.1 Cross-Proofing

Mint will cross-proof from one device to another, and allow individual pages of a document to be viewed or printed. For example, the whole of this document has been cross-proofed on a Perq before being shipped to the Dover. This has allowed me to fine-tune the layout quickly and easily.

3.7.5.2 Error Reporting

Mint includes a facility for reporting bugs to the maintainer. If it recognizes an internal error, it invites the user to create a bug report. This then is shipped automatically to me.

3.7.5.3 Performance

This is not the place to go into a detailed analysis of the performance of Mint, though some figures will be of value to users. There are two issues involved — the performance for small documents, and the way the performance degrades as the size of the document increases. Measurements on version 2A(8) gave the following times per page, and percentages of time spent in swapping, for .Mss files converted all the way to .Press files.

Size = 24 pages.	Time per page = 25.2 secs.	Swapping = 9.3%
Size = 108 pages.	Time per page = 37.0 secs.	Swapping = 17.8%
Size = 152 pages.	Time per page = 36.6 secs.	Swapping = 21.8%
Size = 196 pages.	Time per page = 38.9 secs.	Swapping = 24.6%
Size = 264 pages.	Time per page = 44.9 secs.	Swapping = 27.2%

These are figures for a version with many debugging hooks installed, and with no performance tuning, compiled using a dumb compiler. A guess for the improvement in performance produced by simple tuning is a factor of two; the improvement that might occur if a better compiler were used is less clear, though an additional factor of two seems reasonable.

3.7.6 Examples

The reference section contains many examples of Mint input, and you can browse through it to see what is possible. Also, the files that were used to create the whole of this document are available on line. To retrieve them, use the `update` command, as follows.

```
@update <RETURN>  
Special options?: [n] <RETURN>  
Remote directory: /usr/spice/pos/mint/refman <RETURN>
```

Part Four

Reference Section

The reference section is intended to provide all the information that is required by the aspiring Mint expert. Many of the parts of this section have been culled directly from the listings, so are still in tabular form. The organization of this section is fairly haphazard.

Input conventions	24
Document syntax	25
Galleys	33
Standard galley properties	36
Box environment parameters	46
Units of length	51
Standard values for the environment parameters	52
Slug environments	60
Fonts	63
Colours	67
Computations	70
Box procedures	72
Devices	74
Presentations	75
Standard presentations and printing	78
Layout procedures	78
Macrogenerator	83
Standard macrogenerator facilities	87
Bibliographies	91
Indexes	93
Prefixes and postfixes	96
Counters and labels	98
Standard conversions and counters	102
Miscellaneous layout statements	105
Cross proofing	109
Borders and border styles	110
Mathematical Typesetting	112
DP and Plot	130
Errors	131
Quirks and Oddities	131
Index	****

4.1 Input conventions

Mint allows several front-ends to coexist, and it will take lexemes from which ever one of them is needed for the current environment. At present there are four front-ends: that for Scribe-like text, that for DP, that for Plot, and that for mathematics. It is likely that others will be added shortly. The input conventions for DP and Plot are described in section 4.28, and the input conventions for the mathematical environments is described in section 4.27; this section describes the Scribe-like text input.

As far as has been possible, I have made the Mint input conventions for text the same as those for Scribe; however, there are some differences which have been caused by the very different internal structures of Mint and Scribe. Rather than go into a detailed comparison, I will itemize the principal features of Mint's conventions.

- Environment identifiers (such as `Itemize` and `ovp`), macro identifiers (such as `Include`), and Mint commands (such as `begin`) are case-free, as are the attribute identifiers used to modify environments (such as `Need` and `width`).
- Attributes which take a length as their value must have the units explicitly specified (for example `Need 1 inch`). The parameter identifier can be separated from the value by either a space or an equals. Alas, macro parameter identifiers must be followed by an equals, introducing a regrettable lack of consistency.
- In many situations a blank line is interpreted as completing one environment and starting another of the same type. Generally whether this occurs or not is determined by the syntax of the document, and not by environment parameters, as is the case with Scribe. Since the syntax is not easily accessible to the user, you have to live with what I have chosen; however, other facilities in Mint provide you with similar features.
- If two paragraphs have another environment separating them, and there are no blank lines between, as for example in the case of

```
If two paragraphs have another environment separating them, and there are
no blank lines between, as for example in the case of
@begin(example)
This illustrates the continuation feature
@end(example)
then the last paragraph will be treated as a continuation of the first.
If however a blank line separates the environments, the last paragraph
will start with an indented line (provided that the document type has
indentations for the first line).
```

then the last paragraph will be treated as a continuation of the first. If however a blank line separates the environments, the last paragraph will start with an indented line (provided that the document type has indentations for the first line).

- `Make`, `Enter` and `Begin` are synonyms, as are `Leave` and `End`. Document environments do not need to have an `end`.
- In environments that are filled Mint will expand or contract spaces to fill out the lines. The rate of expansion or contraction is determined by the type of the space, with spaces after the end of a sentence expanding faster than those between words. If a fullstop, exclamation mark or question mark has two or more spaces after it, or is followed by a newline, then the space is regarded as an end-of-sentence space, otherwise as a between-word space. Always start a sentence on a new line,

or precede it by two spaces; follow a fullstop used for an abbreviation with one space only. Spaces after commas, semicolons and colons also expand at different rates. They can be converted to between-word spaces by using @#. Non-expanding spaces are obtained using @w in the same way as Scribe, or by using @ (@ followed by a space).

- Mint does not number pages automatically. See section 4.24.2.1 to find how to include page numbers.
- New environments can be defined in terms of current ones by using `Define`, which is similar to the `Define` and `Equate` commands of Scribe; environments can also be modified by using `Modify`. Both of these statements understand the block structure of a document, and at the end of the environment in which they occur the definitions and modification are popped.

4.2 Document syntax

Mint has a more rigid notion of the syntax of a document than do other document preparation systems. The syntax is enforced by a parser that acts as the front end to the system; this parser performs error-correction, so that in general it is not necessary to specify the complete structure of the document — this can usually be inferred. However, not all general nesting of environments that are possible with Scribe are possible with Mint, so there may be occasions where the structure of a document needs to be changed somewhat. Facilities exist for the disciplined hacker to alter the syntax; see section 4.2.6.

Each document type has a different syntax; however there are similarities between them, and it is convenient to describe the syntactic structure of all document types together.

4.2.1 Syntactic metalanguage

The syntax is described by a simplified version of BNF. A production rule is written as

$$\text{Notion} = [\text{Notion}_1, \text{Notion}_2, \dots, \text{Notion}_n]$$

or

$$\text{Notion} = []$$

The first rule is an abbreviation for the production rules

$$\begin{aligned} \text{Notion} &= \text{Notion}_a^* \\ \text{Notion}_a &= \text{Notion}_1 \mid \text{Notion}_2 \mid \dots \mid \text{Notion}_n \end{aligned}$$

and the second rule specifies that `Notion` generates a terminal string.

In addition to the production rules, the input language to Mint is described by the error-correcting rules. Currently these are somewhat crude (which reflects in the parser occasionally misguessing the correct parse when a trivial amount of analysis would yield it). The error-correction is driven by supplying each

production rule with a set of default `Notions`, from which one is selected if the standard syntax is violated. These defaults are written as

```
{Notion1, Notion2, ... , Notionn},
```

An empty set of defaults implies that the parser will not take corrective action if the parse fails; in general there is at most one default.

4.2.2 Pseudo-Syntactic properties

In addition to the formal syntax, several other properties are associated with the `notions` of a document. `Pack` takes a boolean value, and specifies whether the environment causes typographical display features (such as a blank line, or several consecutive spaces) to be interpreted as a horizontal gap of the appropriate size needed for normal typographical text layout. `Blanklines` takes a value from (`kept`, `ignored`, `invalid`), and specifies whether consecutive blank lines in an environment are to be compressed into a single one or not. The interaction between these two values is somewhat subtle, and will not be described in this version of the document. The other pseudo-syntactic properties are described elsewhere; they are: `dominating`, which is described in section 4.3.5, and `autoincrementing`, which is described in section ???. It was found to be useful to make these values a part of the syntax, because the error-correcting parser uses this contextual information to interpret the meaning the input. For this reason it is not meaningful to change these values during a parse.

4.2.3 Nonterminals

The following are the inbuilt `notions`. They are used to specify the formatting rules that get applied to the input, and they are used in the same way as they are in `Scribe`. For example

```
@begin(example, tabclear, tabset 8ems, tabset 14ems, tabset 22ems)
```

The brief description states informally how they are intended to be used. More details are given in section 4.7.

Abstract	This places a paragraph of text in the position occupied by an abstract.
Align	An environment like <code>verbatim</code> , which treats tabulations in a way that is useful for laying out tables. It is described more fully in section 4.24.4.
Annotation	An annotation is similar to a footnote, though at present it will not appear in the pages.
Appendix	An appendix is similar to a chapter, except for the numbering that is used.
AppendixSection	An appendix section is similar to a section, but is used in appendices.
Article	A document type with numbered sections, subsections, etc., and no table of contents.

Caption	A caption occurs under a figure, or over a table. It is preceded by the figure or table number.
Centre	An environment which centres the text within the margins, breaking the lines at the same places as in the manuscript. It uses the normal roman font.
Chapter	A chapter has a number in front of it; the style depends on the document type. The chapter heading is centred, and uses a large bold typeface.
Commentary	A commentary environment is used to place two pieces of text side-by-side. The left side is in the normal size of font, the right side in a smaller font.
Contents	This is used to create tables of contents. You will never need to use this environment directly.
CopyrightNotice	This places a copyright notice, preceded by the copyright symbol.
Default	The default environment type for all documents except <code>slides</code> . It fills the lines, and uses the normal roman font.
Describe	The describe environment can be used for laying out tables. It is described more fully in section 4.24.5.
Description	This environment produces a list of paragraphs, with the first line outdented. This environment is being used to produce this text.
Display	An environment that narrows the margins, and breaks the lines in the same places as in the manuscript. It uses the normal roman font.
DItem	An <code>ditem</code> is what each of the paragraphs is in an <code>description</code> environment. You usually won't need to use this environment explicitly.
DP	This is used for DP drawings. Section 4.28 describes how to use this environment.
Enumerate	Similar to <code>itemize</code> , except that each paragraph is numbered.
Example	An environment that narrows the margins, and breaks the lines in the same places as in the manuscript. It uses a fixed width font.
Figure	A figure comprises a paragraph of text (using the <code>textpart</code> environment), or a diagram produced by Plot or DP, followed by a caption.
FlushLeft	An environment that narrows its margins, and breaks the lines at the same places as in the manuscript. Lines are flushed up against the left margin. It uses the normal roman font.
FlushRight	Similar the <code>flushleft</code> , except that the lines are flushed up against the right margin.
Foot	This is used for a footnote. The footnote appears at the bottom of the page in which the text that refers to it appears, and it uses a smaller typeface.
Format	An environment that breaks the lines in the same places as in the manuscript. It does not narrow the margins. It uses the normal roman font.
Gloss	A gloss is the right-hand environment of a commentary. You have to explicitly specify an environment as being a <code>gloss</code> , otherwise it will be assumed to be a <code>textpart</code> .

Heading	This produces a centred heading in a bold typeface that is smaller than that used for <code>majorheading</code> .
Item	An <code>item</code> is what each of the paragraphs is in an <code>itemize</code> and <code>enumerate</code> environment. You usually won't need to use this environment explicitly.
Itemize	This produces a list of paragraphs, each preceded by a bullet. (Paragraph is used here in the informal sense, not the sense defined above.)
MajorHeading	This produces a centred heading in a large bold typeface.
Manual	A document type with numbered chapters, sections, subsections, etc., and a table of contents.
Maths	This environment uses special processing of its body to do high-quality mathematical typesetting. Details are given in section 4.27.
Multiple	An environment that groups together other environments within it. It narrows the margins and the line spacing.
Notice	This environment allows you to put other paragraphs on the title page. Their positions are determined by the <code>titlepage</code> environment.
PageCommand	The <code>pagecommand</code> environment is used for several actions concerned with page layout. Normally you will never use this environment directly.
PageHeading	This environment is used for creating page headings. The environments that are nested within it are placed at the top of the pages. It is described in more detail in section 4.24.2.1.
PageFooting	This environment, similar to the <code>pageheading</code> environment, places footings on the pages. It is described in more detail in section 4.24.2.1.
PageOffset	You need this environment if you want to offset the text on pages to the left or right, or up or down. It is described in more detail in section 4.24.2.2.
Paragraph	A paragraph is the lowest level of section heading. It has a number, and is flushed left.
Plot	This environment is used to introduce a diagram produced by the Plot program. Section 4.28 describes how to use this environment.
PrefaceSection	A preface section will start a new page, and produce the heading in a large bold typeface.
ProgramExample	This environment is currently identical to <code>example</code> . In the future it will be used for examples of program text, and will apply conventional program formatting rules.
Quotation	An environment that narrows the margins and line spacing, and fills the lines. It uses the normal roman font.
Report	A document type with numbered chapters, sections, subsections, etc., and a table of contents.
ResearchCredit	This environment places a paragraph of text in the usual position occupied by a research credit.

Section	A section has a number, and is flushed left. It uses the same typeface as that used for chapter .
Slides	A document type without numbered sections, with spacings and fonts suitable for making overhead transparency slides.
SubHeading	A subheading is in a bold typeface, and is flushed left.
SubSection	A subsection has a number, and is flushed left. It uses a bold typeface that is smaller than that used for chapter and section .
Table	A table comprises a caption, followed by a paragraph of text (using the align environment), or a diagram produced using Plot or DP .
Text	The default document type. It has no numbered sections, and has wide-spaced lines.
TextPart	The textpart environment is the left-hand environment of a commentary. Normally you will not need to use this explicitly.
Thesis	A document type with numbered chapters, sections, subsections, etc., and a table of contents.
TitlePage	This environment is used for laying out title pages. It controls the positions of the environments which nest within it.
TitleBox	The environments that are allowed within the title box are majorheading , heading and centre .
Verbatim	This environment narrows the margins, breaks the lines in the same positions as in the manuscript, and uses a fixed width font.
Verse	This environment breaks lines in the same position as they are broken in the manuscript, unless the line is too long to fit within the margins, in which case it indents the broken line.

In addition to these environments, there are the environments **document**, **mfoot**, **mannotation**, **moddsandsods** and **mcontents**, which are inaccessible to you. They control the creation of the galleys.

Note that there is no **equation** environment, as there is in *Scribe*. The **maths** environment plays a similar role.

4.2.4 Production rules common across several document types

In the description below, the nonterminal is followed by the notions it produces, the default notions, the value of **Pack** and the value of **BlankLines**.

The following abbreviations are used.

StdEnvs = Centre, Display, Example, ProgramExample, FlushLeft, FlushRight, Verbatim, Format, Quotation, Align, Default, Itemize, Enumerate, Description, Commentary, Plot, DP, Figure, Table, Verse, Maths, Multiple, Describe

StdTerminals = Centre, Display, Example, ProgramExample, FlushLeft,
FlushRight, example, Format, Quotation, Align, Default,
Plot, DP, Align, Verse, Maths

StdHeadings = MajorHeading, Heading, SubHeading, TitlePage

StdSections = Section, SubSection, Paragraph, Appendix,
AppendixSection, PrefaceSection

StdChapters = Chapter, Section, SubSection, Paragraph, Appendix,
AppendixSection, PrefaceSection

StdTitleEnvs = TitleBox, ResearchCredit, Abstract, CopyrightNotice,
Notice

4.2.4.1 Document environment syntax

Document =	[Text, Report, Thesis, Article, Slides, Manual]			
		{Text}	Skip	Invalid
Text =	[StdEnvs + StdHeadings]	{Default}	Skip	Invalid
Report =	[StdEnvs + StdHeadings + StdChapters]			
		{Default}	Skip	Invalid
Article =	[StdEnvs + StdHeadings + StdSections]			
		{Default}	Skip	Invalid
Thesis =	[StdEnvs + StdHeadings + StdChapters]			
		{Default}	Skip	Invalid
Slides =	[StdEnvs + StdHeadings]	{Verbatim}	Skip	Invalid
Manual =	[StdEnvs + StdHeadings + StdChapters]			
		{Default}	Skip	Invalid

4.2.4.2 Terminal environment syntax

Align =	[[{}	False	Kept
Centre =	[[{}	False	Kept
Default =	[[{}	True	Invalid
Display =	[[{}	False	Kept
DP =	[[{}	True	Ignored
Example =	[[{}	False	Kept
FlushLeft =	[[{}	False	Kept
FlushRight =	[[{}	False	Kept
Format =	[[{}	False	Kept
Maths =	[[{}	False	Kept
Plot =	[[{}	True	Invalid
ProgramExample =	[[{}	False	Kept
Quotation =	[[{}	True	Invalid
Verbatim =	[[{}	False	Kept
Verse =	[[{}	False	Kept

4.2.4.3 Heading environment syntax

MajorHeading =	[[{}	False	Kept
Heading =	[[{}	False	Kept
Subheading =	[[{}	True	Invalid

4.2.4.4 Section environment syntax

PrefaceSection =	[[]]	{}	False	Kept
Section =	[[]]	{}	True	Invalid
SubSection =	[[]]	{}	True	Invalid
Paragraph =	[[]]	{}	True	Invalid
Appendix =	[[]]	{}	True	Invalid
AppendixSection =	[[]]	{}	True	Invalid

4.2.4.5 Chapter environment syntax

PrefaceSection =	[[]]	{}	False	Kept
Chapter =	[[]]	{}	False	Kept
Section =	[[]]	{}	True	Invalid
SubSection =	[[]]	{}	True	Invalid
Paragraph =	[[]]	{}	True	Invalid
Appendix =	[[]]	{}	False	Kept
AppendixSection =	[[]]	{}	True	Invalid

4.2.4.6 Itemization environment syntax

Itemize =	[StdEnvs + Item]	{Item}	Skip	Invalid
Item =	[[]]	{}	True	Invalid
Enumerate =	[StdEnvs + Item]	{Item}	Skip	Invalid
Description =	[StdEnvs + DItem]	{DItem}	Skip	Invalid
Describe =	[StdEnvs]	{FlushLeft}	Skip	Invalid
DItem =	[[]]	{}	True	Invalid
Commentary =	[TextPart, Plot, DP, Gloss]	{TextPart}	Skip	Invalid
TextPart =	[[]]	{}	True	Invalid
Gloss =	[[]]	{}	True	Invalid
Figure =	[TextPart, Plot, DP, Caption]	{TextPart}	Skip	Invalid
Table =	[Align, Plot, DP, Caption]	{Align}	Skip	Invalid
Caption =	[[]]	{}	True	Invalid
Multiple =	[StdEnvs]	{Default}	Skip	Invalid

4.2.4.7 Title page environment syntax

TitlePage =	[StdTitleEnvs]	{Notice}	Skip	Invalid
TitleBox =	[MajorHeading, Heading, Centre]	{Centre}	Skip	Invalid
ResearchCredit =	[StdEnvs]	{Default}	Skip	Invalid
Abstract =	[StdEnvs]	{Default}	Skip	Invalid
CopyrightNotice =	[[]]	{}	True	Invalid
Notice =	[StdEnvs]	{Default}	Skip	Invalid

4.2.4.8 Galley environment syntax

Foot =	[StdEnvs]	{Default}	Skip	Invalid
MFoot =	[Foot]	{Foot}	Skip	Invalid
Annotation =	[StdEnvs]	{Default}	Skip	Invalid
MAnnotation =	[Annotation]	{Annotation}	Skip	Invalid

Contents =	[StdEnvs, StdHeadings]	{Align}	Skip	Invalid
MContents =	[Contents]	{Contents}	Skip	Invalid

4.2.4.9 Page environment syntax

PageHeading =	[StdTerminals]	{Centre}	Skip	Invalid
PageFooting =	[StdTerminals]	{Centre}	Skip	Invalid
PageOffset =	[StdTerminals]	{Centre}	Skip	Invalid
PageCommand =	[]	{ }	True	Invalid
MOddsandSods =	[PageHeading, PageFooting, PageOffset, PageCommand]	{ }	Skip	Invalid

4.2.5 Document Syntax

Each document type has a different syntax. The sets of rules, defaults, and values for `Pack` and `BlankLines` are given in the previous section.

Text Syntax	Terminals + Headings + Items + Pages + Galleys
Report Syntax	Terminals + Headings + Chapters + Items + Title pages + Pages + Galleys
Article Syntax	Terminals + Headings + Sections + Items + Title pages + Pages + Galleys
Thesis Syntax	Terminals + Headings + Chapters + Items + Title pages + Pages + Galleys
Slides Syntax	Terminals + Headings + Items + Pages
Manual Syntax	Terminals + Headings + Chapters + Items + Title pages + Pages + Galleys

4.2.6 Altering the syntax

Sometimes it will be found that the Mint syntax is too restrictive to allow some layouts that you want. For example, you might want an `align` environment in a titlebox, or a commentary in a figure. Mint provides a means of altering the syntax from within the `.mss` file, using the statements `AddRule`, which adds a new right hand side to a production rule; `RemRule`, which removes a right hand side; `AddDefault`, which adds a default to be used by the error correcting parser; and `RemDefault`, which removes a default. Of these, only `AddRule` is likely to be used by any but the most fearless Minter, and even then, I will not be responsible for the havoc that can arise from inappropriate use of the statement.

The form of these statements is

```
@AddRule (LHS, RHS)
@RemRule (LHS, RHS)
@AddDefault (LHS, RHS)
@RemDefault (LHS, RHS)
```

where in all cases the production rules for the LHS are modified by adding or removing the RHS.

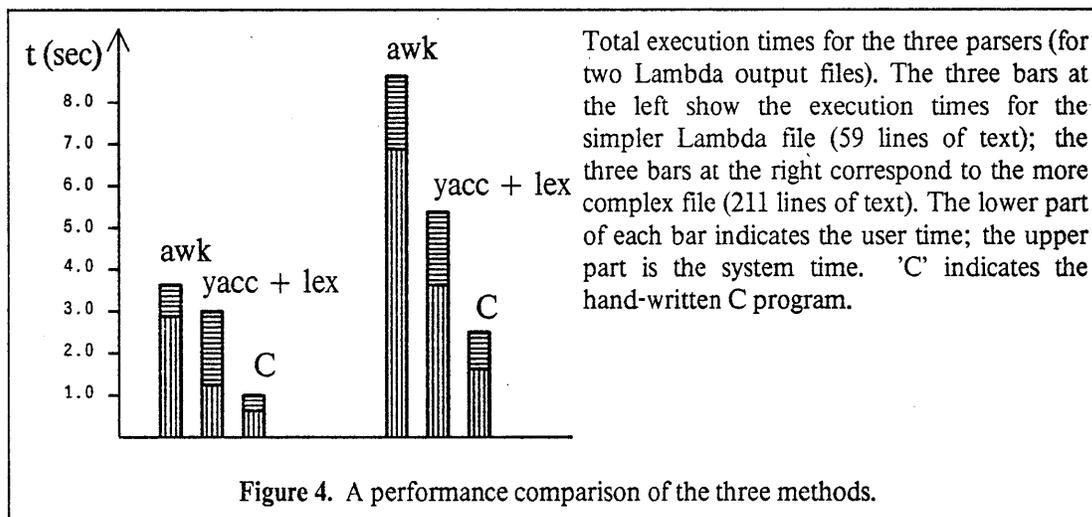
For example, assume that you want to incorporate a `Commentary` environment into a figure. The statements

```
@addrule{figure, commentary)
@begin{figure, width = 6in, borderstyle width1, border = 0.1 in)
@begin[commentary]
@begin[dp, width = 3 in]@include[bargraph.dp]@end[dp]

@begin[gloss, fontsize = n]
Total execution times for the three parsers (for two Lambda output
files). The three bars at the left show the execution times for the
simpler Lambda file (59 lines of text); the three bars at the right
correspond to the more complex file (211 lines of text).
The lower part of each bar indicates the user time; the upper part is the
system time.
'C' indicates the hand-written C program.
@end[gloss]

@end[commentary]
@caption[A performance comparison of the three methods.@label[bargraph]]
@end{figure)
```

produce the following:



4.3 Galleys

Galleys are the principal data structures within Mint. They specify the formatting rules that will be applied for the various environments, they specify the fonts that will be used, and they specify the devices for which the information in the galley will be targetted. In addition to carrying passive information used internally by Mint, and the contents of the manuscript, they also are formatting processes, the interactions between which help create the relationships that exist between pieces of the manuscript — the text and a floating figure, or the text and a footnote, for example.

In the following section I describe the principal properties of galleys; section 4.4 describes the default properties of the galleys set up automatically when Mint starts.

4.3.1 Defining galleys

A new galley may be defined, and its properties specified, from the manuscript file. The statement `NewGalley` creates a new, inactive galley. For example,

```
@NewGalley (FloatingFigures, Dover)
```

creates a new galley with identifier `FloatingFigures` that will contain information formatted for the Dover. To be useful, the galley needs other properties: a *Procedure Family* and a *Font Family*.

4.3.2 Procedure Families

The procedure family associated with a galley determines which information the galley will receive from the manuscript, and how it will format the information. Basically the procedure family of a galley specifies a top-down parser and semantic analyser that takes the output of the error-correcting parser, and performs the transformations on it necessary to produce the formatted text. Whether a galley receives the output from the error-correcting parser is determined by two factors — whether the galley has a formatting procedure for the particular environment being currently treated by the error-correcting parser, and whether a *Dominating Environment* has excluded the galley or not. See section 4.3.5 for a description of dominating environments.

The procedure family of a galley is maintained as a vector, indexed by the environment identifier — `Itemize`, `Caption`, etc. An entry in the vector comprises two parts, a *Procedure Indicator* and an *Environment Indicator*. The procedure indicator specifies which (recursive) procedure will be used to parse and analyse the input; the environment indicator specifies which set of box environment parameters will be used for this analysis. There are fewer procedures than environment parameter sets, because several environments can be parsed using the same procedure. For example, analysing the `FlushLeft`, the `Verbatim` and the `Gloss` environments is done by the same procedure, because the differences between these environments are captured completely by the differences in the environment parameters. See section 4.12 for a description of the procedures, and see section 4.5 for a description of the box environment parameters.

A procedure is associated with some entry in the procedure family vector by the statement `AssocProc`, which takes the name of a galley, the external identifier of the environment, the identifier of the procedure, and the identifier of the environment parameters. For example

```
@AssocProc (FloatingFigures, DP, BoxStandard0, EnvDP0)
```

specifies that the `FloatingFigures` galley is able to handle the DP environment, using the procedure

`BoxStandard0` and the environment parameters `EnvDP0`. If the entry was already occupied, it is overwritten.

A word of caution. By placing entries in the procedure vector, you are (meta-) programming a parser and semantic analyser. There are many ways that you can make mistakes, and there are only a limited number of checks it is reasonable for me to do to help you out. You are probably in good shape adding terminal environments, though.

4.3.3 Font Families

A font family is a two dimensional array of *Font Indicators* that indicates the fonts to use in the box environments. Font Families are described in more detail in section 4.9, which explains how to add fonts to the font family, and how to manufacture fonts from existing ones.

When a galley has been created, using `NewGalley`, the font family is empty. Fonts should be associated with the family before the galley receives input, otherwise Mint will use a default font.

4.3.4 Installing a galley

In addition to maintaining passive information, a galley is an active process, which has associated with it an execution context and the stack of activation records of the procedures of the parser and analyser that have been entered and not yet left. When a galley is first created, it has no execution context, and its stack is empty. A galley is given a context and made to play an active role in formatting by *installing* it, using the statement `InstallGalley` (basically, the galley is placed on the Mint scheduler queue). `InstallGalley` takes two parameters: the name of the galley, and the name of the procedure family entry that is at the bottom of the execution stack for the galley.

```
. @InstallGalley (FloatingFigure, DP)
```

The galley will be suspended, and will become active only when the error-correcting parser finds the appropriate environment in the manuscript. Thereafter the error-correcting parser will feed information to the galley until the environment ends, at which time the galley will suspend. In order to be useful, therefore, the lowest invocation in the galley's execution context should be a procedure that loops, and calls other procedures to format specific environments. Because the error-correcting parser handles the input first, the looping procedure need not have an explicit environment identifier available to the user. More details are beyond the scope of the reference manual; but see how the Footnotes galley is constructed for a model.

4.3.5 Dominating environments

As mentioned above, two factors determine whether a galley will receive the output from the error-correcting parser: first, whether the galley has a procedure for the current environment, and second,

whether a *dominating environment* has excluded other galleys. It is necessary to be able to exclude galleys that are eligible by the first rule, since many galleys may be able to format a `default` environment, for example; however, if this environment occurs within a footnote, only the galleys handling footnotes should receive the output of the error-correcting parser.

In order to restrict the set of galleys, environments can be made *dominating*. If a dominating environment is current, then only those galleys that have the dominating environment in their procedure family will receive input, and all the others will remain blocked until the environment terminates. The specification of whether an environment is dominating or not is made during the specification of the syntax. The following environments are specified as dominating in the current system.

Foot	Annotation	PageHeading	PageFooting
PageOffset	PageCommand		

4.4 Standard Galley Properties

This section describes the properties of the standard galleys that are made available by default in the current version of Mint. In order to simplify the description, I introduce several collections of procedure families, font families, prefixes and styles before showing in section 4.4.5 how each of the galleys is composed. See section 4.3 for a description of procedure families and font families, section 4.21 for a description of prefixes, and section 4.4.4 for a description of galley styles.

The current version of Mint uses five galleys — the `main` galley, into which most of the document is placed; the `footnotes` galley, which is used to receive footnotes; the `annotations` galley, which receives annotations; the `oddsandsods` galley, which receives page layout information; and the `contents` galley which receives the table of contents. At present DP and Plot drawings go into the main galley.

4.4.1 Procedure families

The following procedure collections are defined. The name of the environment is followed by the procedure indicator and environment indicator that the environment uses. See section 4.3.2 for the meanings of these terms.

4.4.1.1 Terminal procedures

Align	BoxStandard0	EnvAlign0
Centre	BoxStandard0	EnvCentre0
Default	BoxStandard0	EnvDefault0
Display	BoxStandard0	EnvDisplay0
DP	BoxStandard0	EnvDP0
Example	BoxStandard0	EnvExample0
FlushLeft	BoxStandard0	EnvFlushLeft0
FlushRight	BoxStandard0	EnvFlushRight0

Format	BoxStandard0	EnvFormat0
Maths	BoxMaths0	EnvMaths0
Plot	BoxStandard0	EnvPlot0
ProgramExample	BoxStandard0	EnvProgramEx0
Quotation	BoxStandard0	EnvQuotation0
Verbatim	BoxStandard0	EnvVerbatim0
Verse	BoxStandard0	EnvVerse0

4.4.1.2 Heading procedures

Heading	BoxStandard0	EnvHeading0
MajorHeading	BoxStandard0	EnvMajorHeading0
SubHeading	BoxStandard0	EnvSubHeading0

4.4.1.3 Chapter procedures

Appendix	BoxSectionEnv0	EnvAppendix0
AppendixSection	BoxSectionEnv0	EnvAppendixSec0
Chapter	BoxSectionEnv0	EnvChapter0
Paragraph	BoxSectionEnv0	EnvParagraph0
PrefaceSection	BoxStandard0	EnvPrefaceSec0
Section	BoxSectionEnv0	EnvSection0
SubSection	BoxSectionEnv0	EnvSubSection0

4.4.1.4 Section procedures

Appendix	BoxSectionEnv0	EnvAppendix1
AppendixSection	BoxSectionEnv0	EnvAppendixSec1
Paragraph	BoxSectionEnv0	EnvParagraph0
PrefaceSection	BoxStandard0	EnvPrefaceSec0
Section	BoxSectionEnv0	EnvSection0
SubSection	BoxSectionEnv0	EnvSubSection0

4.4.1.5 Itemization procedures

Caption	BoxCaption0	EnvCaption0
Commentary	BoxCommentary0	EnvCommentary0
Describe	BoxDescribe0	EnvDescribe0
Description	BoxMultiple0	EnvDescription0
DItem	BoxStandard0	EnvDItem0
Enumerate	BoxEnumerate0	EnvEnumerate0
Figure	BoxFigure0	EnvFigure0
Gloss	BoxStandard0	EnvGloss0
Item	BoxStandard0	EnvItem0
Itemize	BoxItemize0	EnvItemize0
Multiple	BoxMultiple0	EnvMultiple0
TextPart	BoxStandard0	EnvTextPart0
Table	BoxTable0	EnvTable0

4.4.1.6 Title page procedures

Abstract	BoxMultiple0	EnvAbstract0
CopyrightNotice	BoxSectionEnv0	EnvCopyrightNO
Notice	BoxMultiple0	EnvNotice0
ResearchCredit	BoxMultiple0	EnvResearchCr0

TitlePage	BoxMultiple0	EnvTitlePage0
TitleBox	BoxMultiple0	EnvTitleBox0

4.4.1.7 Document type procedures

Article	BoxDocType0	EnvArticle0
Document	BoxDocument0	EnvDocument0
Letter	BoxDocType0	EnvLetter0
Manual	BoxDocType0	EnvManual0
Report	BoxDocType0	EnvReport0
Slides	BoxDocType0	EnvSlides0
Text	BoxText0	EnvText0
Thesis	BoxDocType0	EnvThesis0

4.4.1.8 Footnote procedures

Foot	BoxCrossRef0	EnvFoot0
MFoot	BoxGalley0	EnvMFoot0

4.4.1.9 Annotation procedures

Annotation	BoxCrossRef0	EnvAnnotation0
MAnnotation	BoxGalley0	EnvMAnnotation0

4.4.1.10 Contents procedures

Contents	BoxMultiple0	EnvContents0
MContents	BoxGalley0	EnvMContents0

4.4.1.11 Miscellaneous procedures

MOddsandSods	BoxGalley0	EnvMOddsandSods0
PageCommand	BoxCRTerm0	EnvPageCommand0
PageHeading	BoxCrossRef0	EnvPageHeading0
PageFooting	BoxCrossRef0	EnvPageFooting0
PageOffset	BoxCrossRef0	EnvPageOffset0

4.4.2 Font families

The following font collections are defined.

Table 1. Perq fonts 0 (Main galley)					
Face code	Font size				
	ss	s	n	l	ll
r	-	Gacha9	TimesRoman12	-	-
i	-	-	TimesRoman12i	-	-
b	-	Gacha9	TimesRoman12b	TimesRoman14	TimesRoman18
c	-	-	-	-	-
g	-	-	-	-	-
t	-	Gacha9	Gacha12	-	-
p	-	-	-	-	-
z	-	-	-	-	-

Table 2. Dover fonts 0 (Main galley)					
Face code	Font size				
	ss	s	n	l	ll
r	TimesRoman6	TimesRoman8	TimesRoman10	TimesRoman11	TimesRoman14
i	-	TimesRoman8i	TimesRoman10i	-	-
b	-	TimesRoman8b	TimesRoman10b	TimesRoman11b	TimesRoman14b
c	-	-	CapsFont ^{a)}	-	-
g	-	-	Hippo10	-	-
t	-	Gacha8	Gacha10	-	-
p	-	-	TimesRoman10bi	-	-
z	-	-	ZFont10	-	-

a) This is constructed from TimesRoman10 and TimesRoman8.

Table 3. Perq fonts 1 (Annotations and Footnotes galleys)					
Face code	Font size				
	ss	s	n	l	ll
r	-	-	Gacha9	-	-
i	-	-	-	-	-
b	-	-	-	-	-
c	-	-	-	-	-
g	-	-	-	-	-
t	-	-	-	-	-
p	-	-	-	-	-
z	-	-	-	-	-

Table 4. Dover fonts 1 (Annotations and Footnotes galleys)					
Face code	Font size				
	ss	s	n	l	ll
r	-	TimesRoman6	TimesRoman8	-	-
i	-	TimesRoman6i	TimesRoman8i	-	-
b	-	TimesRoman6b	TimesRoman8b	-	-
c	-	-	-	-	-
g	-	-	-	-	-
t	-	-	-	-	-
p	-	-	-	-	-
z	-	-	-	-	-

Table 5. Perq fonts 2 (OddsandSods galley)

Face code	Font size				
	ss	s	n	l	ll
r	-	Gacha9	TimesRoman12	-	-
i	-	-	TimesRoman12i	-	-
b	-	-	TimesRoman12b	-	-
c	-	-	-	-	-
g	-	-	-	-	-
t	-	-	Gacha12	-	-
p	-	-	-	-	-
z	-	-	-	-	-

Table 6. Dover fonts 2 (OddsandSods galley)

Face code	Font size				
	ss	s	n	l	ll
r	-	TimesRoman8	TimesRoman10	-	-
i	-	-	TimesRoman10i	-	-
b	-	-	TimesRoman10b	-	-
c	-	-	-	-	-
g	-	-	-	-	-
t	-	-	-	-	-
p	-	-	-	-	-
z	-	-	-	-	-

Table 7. Perq fonts 3 (Slides Main galley)

Face code	Font size				
	ss	s	n	l	ll
r	TimesRoman12	TimesRoman12	TimesRoman14	TimesRoman18	TimesRoman18
i	-	-	-	-	-
b	-	Gacha9	TimesRoman12b	TimesRoman14	TimesRoman18
c	-	-	-	-	-
g	-	-	-	-	-
t	-	-	TimesRoman14	-	-
p	-	-	-	-	-
z	-	-	-	-	-

Table 8. Dover fonts 3 (Slides Main galley)

Face code	Font size				
	ss	s	n	l	ll
r	Helvetica12	Helvetica14	Helvetica18	-	-
i	-	Helvetica14i	Helvetica18i	-	-
b	-	Helvetica14b	Helvetica18b	Helveticad24	Helveticad30
c	-	-	-	-	-
g	-	-	-	-	-
t	-	-	Helvetica18	-	-
p	-	-	-	-	-
z	-	-	-	-	-

Table 9. Perq fonts 4 (Slides OddsandSods galley)

Face code	Font size				
	ss	s	n	l	ll
r	TimesRoman12	TimesRoman12	TimesRoman14	TimesRoman18	TimesRoman18
i	-	-	-	-	-
b	-	-	-	-	-
c	-	-	-	-	-
g	-	-	-	-	-
t	-	-	-	-	-
p	-	-	-	-	-
z	-	-	-	-	-

Table 10. Dover fonts 4 (Slides OddsandSods galley)

Face code	Font size				
	ss	s	n	l	ll
r	Helvetica12	Helvetica12	Helvetica14	Helvetica18	Helvetica18
i	-	-	-	-	-
b	-	-	-	-	-
c	-	-	-	-	-
g	-	-	-	-	-
t	-	-	-	-	-
p	-	-	-	-	-
z	-	-	-	-	-

4.4.3 Prefixes and postfixes

These are collected into the following. See section 4.21 for a description of these prefixes and postfixes.

4.4.3.1 Prefixes0Chapters

Chapter	PrefixChapter0	Section	PrefixSection0
SubSection	PrefixSubSection0	Paragraph	PrefixParagraph0
Appendix	PrefixAppendix0	AppendixSection	PrefixAppendixSec0

4.4.3.2 Prefixes1Chapters

These are the same as for Prefixes0Chapters, expect that the Chapter prefix is PrefixChapter1.

4.4.3.3 Prefixes0Sections

Section	PrefixSection0	SubSection	PrefixSubSection0
Paragraph	PrefixParagraph0	Appendix	PrefixAppendix1
AppendixSection	PrefixAppendixSec0		

4.4.3.4 Prefixes0Items

Figure PrefixFigure0 Table PrefixTable0

4.4.3.5 Prefixes0TitleEnvs

CopyrightNotice PrefixCopyrtN0

4.4.3.6 Postfixes0Terminals

Maths PostfixEqn0

4.4.4 Standard styles

Styles are parameters that control the general appearance of a document, and are set by default, or altered when the document is made. See section 4.5.2.1 for a description of their actions. The following two collections of style parameters are specified. Let $XWidth$ be the width of the page on the target device, and $YHeight$ be the height of the page. Also let $XInc$ be equal to $XWidth/40$, and $YInc$ be equal to $YHeight/50$. The style parameters are as follows.

Styles0			
Width	$XWidth*3/4$	RAbove	YInc
RLM	0	NAbove	YInc div 2
NLM	XInc	RBelow	YInc
RRM	0	NBelow	YInc div 2
NRM	XInc	RGap	YInc div 4
Indent	XInc	NGap	YInc div 10
JustifyLeft	True	HGap	YInc div 10
JustifyLeftLast	True	JustifyRight	True
ImageColour	Black	JustifyRightLast	False
RasterFunction	R0r	BackgroundColour	Transparent
Styles1			
Width	$XWidth*3/4$	RAbove	YInc * 2
RLM	0	NAbove	YInc
NLM	XInc	RBelow	YInc * 2
RRM	0	NBelow	YInc
NRM	XInc	RGap	YInc div 2
Indent	0	NGap	YInc div 3
JustifyLeft	True	HGap	YInc div 10
JustifyLeftLast	True	JustifyRight	True
ImageColour	Black	JustifyRightLast	False
RasterFunction	R0r	BackgroundColour	Transparent

4.4.5 The galley parameters for the document types

4.4.5.1 Text, form 0

The following galleys, prefixes and postfixes are defined.

Main galley, invoked with Document
Fonts: Fonts0

Procedures: Terminals + Headings + Itemizations + DocTypes
Styles: Styles1

FootNotes galley, invoked with MFoot
Fonts: Fonts1
Procedures: Terminals + Itemizations + FootNotes
Styles: Styles0

Annotations galley, invoked with MAnnotation
Fonts: Fonts1
Procedures: Terminals + Itemizations + Annotations
Styles: Styles0

OddsandSods galley, invoked with MOddsandSods
Fonts: Fonts2
Procedures: Terminals + Miscellaneous
Styles: Styles1

Prefixes and postfixes
Prefixes0Items + Postfixes0Terminals

4.4.5.2 Text, form 1

Document type Text, form 1 is identical to Text, form 0 except that Styles0 is used for the main galley, instead of Styles1.

4.4.5.3 Report, form 0

The following galleys, prefixes and postfixes are defined.

Main galley, invoked with Document
Fonts: Fonts0
Procedures: Terminals + Headings + Chapters + Itemizations + TitleEnvs
Styles: Styles0

FootNotes galley, invoked with MFoot
Fonts: Fonts1
Procedures: Terminals + Itemizations + FootNotes
Styles: Styles0

Annotations galley, invoked with MAnnotation
Fonts: Fonts1
Procedures: Terminals + Itemizations + Annotations
Styles: Styles0

OddsandSods galley, invoked with MOddsandSods
Fonts: Fonts2
Procedures: Terminals + Miscellaneous
Styles: Styles0

Contents galley, automatically invoked
Fonts: Fonts0
Procedures: Terminals + Headings + Itemizations + Contents
Styles: Styles0

Prefixes and postfixes
Prefixes0Chapters + Prefixes0Items + Prefixes0TitleEnvs + Postfixes0Terminals

4.4.5.4 Article, form 0

The following galleys, prefixes and postfixes are defined.

Main galley, invoked with Document

Fonts: Fonts0
Procedures: Terminals + Headings + Sections + Itemizations + TitleEnvs
Styles: Styles0

FootNotes galley, invoked with MFoot

Fonts: Fonts1
Procedures: Terminals + Itemizations + FootNotes
Styles: Styles0

Annotations galley, invoked with MAnnotation

Fonts: Fonts1
Procedures: Terminals + Itemizations + Annotations
Styles: Styles0

OddsandSods galley, invoked with MOddsandSods

Fonts: Fonts2
Procedures: Terminals + Miscellaneous
Styles: Styles0

Prefixes and postfixes

Prefixes0Sections + Prefixes0Items + Prefixes0TitleEnvs + Postfixes0Terminals

4.4.5.5 Thesis, form 0

The following galleys, prefixes and postfixes are defined.

Main galley, invoked with Document

Fonts: Fonts0
Procedures: Terminals + Headings + Chapters + Itemizations + TitleEnvs
Styles: Styles0

FootNotes galley, invoked with MFoot

Fonts: Fonts1
Procedures: Terminals + Itemizations + FootNotes
Styles: Styles0

Annotations galley, invoked with MAnnotation

Fonts: Fonts1
Procedures: Terminals + Itemizations + Annotations
Styles: Styles0

OddsandSods galley, invoked with MOddsandSods

Fonts: Fonts2
Procedures: Terminals + Miscellaneous
Styles: Styles0

Contents galley, automatically invoked

Fonts: Fonts0
Procedures: Terminals + Headings + Itemizations + Contents
Styles: Styles0

Prefixes and postfixes

Prefixes0Chapters + Prefixes0Items + Prefixes0TitleEnvs + Postfixes0Terminals

4.4.5.6 Slides, form 0

The following galleys, prefixes and postfixes are defined.

Main galley, invoked with Document

Fonts: Fonts3

Procedures: Terminals + Headings + Itemizations

Styles: Styles1

OddsandSods galley, invoked with MOddsandSods

Fonts: Fonts4

Procedures: Terminals + Miscellaneous

Styles: Styles1

Prefixes and postfixes

Prefixes0Items + Postfixes0Terminals

4.4.5.7 Manual, form 0

The following galleys, prefixes and postfixes are defined.

Main galley, invoked with Document

Fonts: Fonts0

Procedures: Terminals + Headings + Chapters + Itemizations + TitleEnvs

Styles: Styles0

FootNotes galley, invoked with MFoot

Fonts: Fonts1

Procedures: Terminals + Itemizations + FootNotes

Styles: Styles0

Annotations galley, invoked with MAnnotation

Fonts: Fonts1

Procedures: Terminals + Itemizations + Annotations

Styles: Styles0

OddsandSods galley, invoked with MOddsandSods

Fonts: Fonts2

Procedures: Terminals + Miscellaneous

Styles: Styles0

Contents galley, automatically invoked

Fonts: Fonts0

Procedures: Terminals + Headings + Itemizations + Contents

Styles: Styles0

Prefixes and postfixes

Prefixes0Chapters + Prefixes0Items + Prefixes0TitleEnvs + Postfixes0Terminals

4.4.5.8 Manual, form 1

Document type Manual, form 1 is identical to Manual, form 0 except that Prefixes1Chapters is used for the main galley, instead of Prefixes0Chapters.

4.5 Box Environment Parameters

The attributes of a box determine the appearance of the box and the way that the information in the box is laid out. The attributes are set to standard values on entering an environment, but they may be changed using environment parameters, or globally using `define` and `modify`. Several related environments, such as `centre`, `flushleft`, etc., differ only because they use different values of some of the environment parameters. For a complete list of the standard values for all the environments, see section 4.7. In general the environment parameters do not affect the appearance of a box directly, but instead they do it by being parameters to *computations*, which generate the values needed by the low level Mint formatting routines. Since the Minter has control over which computations will get applied, it is a little misleading to say, for example, that the `width` parameter specifies the width of a box. However, there is usually a direct relation between an environment parameter and the value generated for the low level routines, so this distinction can be ignored by the casual user. In the description below I am assuming the the standard computations will be applied.

Environment parameters are of two classes: the standard ones, which are defined for all the environments (though they are not necessarily always meaningful), and additional environment parameters, which are usually specific to a particular environment.

4.5.1 Standard attributes

The following are the standard attributes, and the type of the values they may be set to. See section 4.6 for a description of these types, and how values of them are represented.

Width	<p>This defines the width of the external dimensions of the box that encloses the information in an environment. See figure *** for the meaning of these terms. The width is specified in units of horizontal length, for example</p> <pre>@begin(plot, width 3.25 inches)</pre> <p>Note that the height of a box relative to its width is determined by its contents.</p>
Above, Below	<p>These specify the minimum space above and below a box (this statement applies to the standard environments; see section 4.11 for more details on this). The gap between two boxes, <i>A</i> followed by <i>B</i> is computed as the maximum of the <code>below</code> of <i>A</i> and the <code>above</code> of <i>B</i>. The values of <code>above</code> and <code>below</code> are specified in units of vertical length, e.g.</p> <pre>@begin(Heading, Above = 3 lines, Below = 1.3 lines)</pre>
LeftMargin	<p>This specifies the horizontal distance of the slug's left margin from the internal dimension of the box. It is specified in horizontal length units, e.g.</p> <pre>@begin(itemize, leftmargin = 5.5 ems)</pre>
ExtraLeftMargin	<p>This specifies the horizontal distance of the slug's left margin from the internal dimension of the box for continuation lines in the <code>verse</code> environment. It is specified in horizontal length units, e.g.</p>

	<code>@begin(verse, extraleftmargin = 4 quads)</code>
RightMargin	This specifies the horizontal distance of the slug's right margin from the internal dimension of the box. It is specified in horizontal length units, e.g. <code>@begin(itemize, rightmargin = 20 picas)</code>
Continue	This determines how the computations for the margins should be performed in the case where two boxes have a third box of a different kind between them. If this parameter has the value <code>allow</code> , and there are no blank lines separating the three boxes, then the margin computations for the third box are performed as though the third box is a continuation of the first; if the value is <code>disallow</code> , then the margins of the third box are computed independently of the context in which it occurs; and if it is <code>true</code> the margins are treated as if the box is a continuation of the previous box of this kind.
Gap	This parameter specifies the gap that will be placed between the slugs in the box. It takes units that are vertical lengths.
JustifyLeft	And <code>JustifyRight</code> , <code>JustifyLeftLast</code> , <code>JustifyRightLast</code> . These specify the nature of the justification actions taken when formatting text slugs. <code>JustifyLeftLast</code> and <code>JustifyRightLast</code> control the justification of the last line of a box; <code>JustifyLeft</code> and <code>JustifyRight</code> control the justification of the other lines in the box. They take values from (<code>True</code> , <code>False</code>). Suitable combinations of these values are used to produce most of the different box formats; for example, if all are <code>false</code> the information in a box is centred, if <code>JustifyLeft</code> and <code>JustifyRight</code> only are <code>true</code> , then the information in the box is flushed left.
ImageColour	This determines the colour of the image; that is, the colour of the characters which are put into boxes or the lines that are drawn in diagrams. See section 4.10 for a description of how to use this parameter.
BackgroundColour	<code>BackgroundColour</code> determines the colour of the background of a box. See section 4.10 for a description of how to use this parameter.
FontSize	This determines the default font size used for the fonts in the box. It takes values from (<code>11</code> , <code>1</code> , <code>n</code> , <code>s</code> , <code>ss</code>). See section 4.8.2 for the meanings of these values.
FaceCode	This specifies the default face code of the box. It takes values from (<code>r</code> , <code>i</code> , <code>b</code> , <code>c</code> , <code>g</code> , <code>t</code> , <code>p</code> , <code>z</code> , <code>m0</code> , <code>m1</code> , <code>m2</code> , <code>f0</code> , ..., <code>f1</code>). See sections 4.8.1 and 4.8.3 for the meaning of these values.
UnderLine	This determines whether slugs in the box will be underlined. It takes values from the set (<code>None</code> , <code>NonBlank</code> , <code>All</code> , <code>Alphanumeric</code> , <code>EraseNonBlank</code> , <code>EraseAll</code> , <code>EraseAlphanumeric</code> , <code>OverNonBlank</code> , <code>OverAll</code> , <code>OverAlphanumeric</code>). Due to an oversight, it is only possible to set one of these values, rather than several non-conflicting ones. It will get fixed.
RasterFunction	This determines the raster function used to draw the characters in the slugs. It can assume values from the set specified in the device properties, see section 4.13.
PageStyle	This determines the page style that will be used to display the contents of the box when pages are made. It takes values from (<code>Skip</code> , <code>Default</code> , <code>TitlePage</code>). See section 4.14 for more details on how choices of this parameter affect the page layout.

Need	The <i>need</i> of a box is the amount of space that should be available at the bottom of a page area for slugs from this box to be placed in the page; if the space remaining is less than this amount, a new page is started. It takes a signed vertical distance as its value, or it takes the value <code>group</code> , which is interpreted to mean that the box must not be spilt across pages. (But no <code>group</code> yet).
Border	This determines the width of the border that exists between the outer edge of the box and the inner edge; see figure *** for a clarification of these terms. It takes values that are distances.
BorderStyle	The border style determines the appearance of the border drawn around the box. Two border styles are predefined — <code>NoBorder</code> , which leaves the border empty, and <code>Width1</code> which draws a border of width 1/100 th inch around the box. See section 4.26 for more details about creating border styles.
CompLM	This determines the <i>computation</i> that will be used to obtain the left margin of the slugs. Several standard computations are provided to handle the normal cases of indented text, verse, etc., but the hacker can also go in and provide his own, for all sorts of exotic purposes. The parameter takes small integer values. See section 4.11 for more details.
CompRM	In a similar way, this specifies the computation for the right margin.
CompGap	This specifies the computation for the gaps separating the slugs.
CompXPosn	This specifies the computation to be used to determine the <i>X</i> -position of the box relative to its neighbours; that is, its horizontal position in the galley.
CompYPosn	This specifies the computation to be used to determine the <i>Y</i> -position of the box relative to its neighbours; that is, its vertical position in the galley.
CompWidth	This specifies the computation to determine how the width of the box is derived.
CompFont	This specifies the font for each of the slugs in the box.
TabSet	This parameter takes a horizontal distance, and sets a tabulation at that point (measured from the exterior of the box). This parameter can be used several times for a single box; each use sets a different tabulation. Up to 10 tabulations can be set.
TabClear	This does not take a value. It clears all the tabulations that have been set so far. Since the parameters are processed from left to right, the <code>TabClear</code> should precede any uses of <code>TabSet</code> .
TabDivide	This takes an integer value. It sets up that number of equidistant tabulations across the inner border of the box, with the last one on the right inner border. Any previous tabulations are destroyed.

4.5.2 Additional parameters

Additional parameters to those above may be passed to an environment. Facilities exist, buried within Mint, to extract the parameter identifiers and their values, and act upon them. If an additional parameter has not been used at the end of the environment, an error message is produced. Thus misspelled attributes will be indicated at the end, rather than the beginning, of an environment.

Several of the standard environments take additional parameters — the document type environments, the `maths` environment, and the `TitlePage` environment.

4.5.2.1 Style parameters for document types

The following parameters alter the values of an additional set of environment parameters that are associated with each box, and from which the default values of the attributes above are sometimes derived. Since these additional parameters are only interpreted by the document type environments, they have been designed to cause global changes in the “style” of the document. They are termed *style parameters*. If you wish to change the appearance of a document, you should change these parameters, since it is from these that the box environment parameters are inherited. It is no use changing, for example, the `gap` parameter of a document if you want to change the inter-line gap. Change the value of `RGap` or `NGap`.

RLM	This sets the size of the regular left margin of the document. It takes a horizontal distance for its value.
NLM	This sets the size of the narrow left margin of the document. It takes a horizontal distance for its value.
RRM	This sets the size of the regular right margin of the document. It takes a horizontal distance for its value.
NRM	This sets the size of the narrow right margin of the document. It takes a horizontal distance for its value.
RAbove	This sets the size of the regular above leading. It takes a vertical distance for its value.
NAbove	This sets the size of the narrow above leading. It takes a vertical distance for its value.
RBelow	This sets the size of the regular below leading. It takes a vertical distance for its value.
NBelow	This sets the size of the narrow below leading. It takes a vertical distance for its value.
RGap	This sets the size of the regular gap leading. It takes a vertical distance for its value.
NGap	This sets the size of the narrow gap leading. It takes a vertical distance for its value.
HGap	This sets the size of the “heading gap”, an extra gap used to compute the gaps for heading and section environments. It takes a vertical distance for its value.
Indent	This sets the size of the indent, used by the <code>CompLM</code> computations to determine the indentation for the first lines of boxes. It takes a horizontal distance for its value.
Filler	This sets the <i>filler lexeme</i> that is placed in a slug in place of a cross reference to a label should the label not yet be defined. It takes an arbitrary string.

CiteStyle	This sets the citation style for the document. It takes the value <code>stdnumeric</code> , <code>stdalphabetic</code> , <code>cacm</code> or <code>ieee</code> ; its default value is <code>stdnumeric</code> . See section <code>bibs</code> for a description of the bibliography facility.
IndexStyle	This sets the style in which the index will appear. It take the value <code>macro</code> or <code>style1</code> ; the default is <code>style1</code> . See section 4.20 for a description of the index facility.

4.5.2.2 Additional parameters for title pages

The `TitlePage` environment takes a set of additional parameters that allow the default positions of the various boxes on the title page to be changed. See section 4.16.3.2 for the way in which the title page layout procedure works. All these additional parameters take vertical distances for their values — these are the distances of the top external border of the corresponding box from the top of the page. The default values are shown in parenthesis, and are measured in units of `PageHeight` for the target device. Unfortunately, due to an oversight, you cannot use `pageheight` units if you want to set these yourself, you must use one of the other vertical units.

TitleBox	This specifies the position of the title box. Inside the title box other environments may be nested — the relative positions of these environments are not altered (0.18).
ResearchCredit	This specifies the position of the research credit box (0.80).
CopyRightNotice	This specifies the position of the copyright notice box (0.75).
Abstract	This specifies the position of the slug that is created with the heading; a suitable gap is inserted to separate it from the abstract box (0.55).
Notice	This specifies where the (first or only) notice will appear (0.39).
Notice1... Notice6	These specify where the corresponding notices will appear, counting in lexicographic order of appearance of the notices in the original manuscript.

4.5.2.3 Itemize and Enumerate

The `itemize` and `enumerate` environments take two parameters that allow you to control the position of the margin against which the bullets and counters are flushed, and the width of the box in which the bullets and counters are placed.

BulletPosn	This specifies the horizontal distance of the bullets or numbers from the left margin. Its default value is 0.05 of the width of the box.
LeftMPosn	This specifies the horizontal distance of the left margin of the text in an <code>itemize</code> and <code>enumerate</code> environment. Its default value is 0.0625 of the width of the box.

4.5.2.4 Maths

The `maths` environment takes a number of additional parameters. These are described in section 4.27.

4.6 Units of length

Unlike Scribe, Mint has a fairly precise notion of units of length, and will require you to specify the units wherever a distance is required. There are three classes of units of length — *absolute units*, measured in lengths of platinum, lengths of Saxon kings' beards, etc; *device relative* units, measured in units such as rasters; and *layout relative* units, measured in terms of the sizes of pages, or characters in some font, etc.

Because many devices have different resolutions in the horizontal and vertical directions, Mint classifies device relative and layout relative units into horizontal and vertical measures. Wherever it says, in this manual, that “Mint expects a vertical unit” then Mint will require the unit of length to be an appropriate vertical unit. In addition, during galley creation it is not appropriate to talk about the size of a page, and during page layout there is no default font to use for deriving the size of a character. Mint checks that the appropriate relative units are being used. (Actually, Mint isn't quite consistent here, and this part of the implementation needs to be tidied up. You will be safe using absolute units everywhere.)

The units are as follows.

4.6.1 Absolute units

The units are expressed in ratios relative to one inch (remember the Saxon king?). They may be used for both horizontal and vertical distances.

<code>inch</code>	Also <code>inches</code> , <code>in</code> , <code>ins</code> . Just as you would expect.
<code>point</code>	Also <code>points</code> . Printers' units of measure, appropriate for fonts. 1 point = 0.013837 inches.
<code>pica</code>	Also <code>picas</code> . Ditto. 1 pica = 0.166044 inch.
<code>cm</code>	Also <code>cms</code> . 1 cm = 0.3937 inch.
<code>micas</code>	Also <code>micas</code> . 1 mica = 0.001 cm = 0.0003937 inch.

4.6.2 Raster lengths

Mint assumes that there is some device-dependent unit of distance, the *raster*, which may not necessarily be the same in absolute units in the *x* and *y* directions. Mint uses raster units internally, and converts all other units into raster units. In fact the size of the raster is just some convenient unit for the device; it may have no relation to the way the device lays down its image (and in particular, the raster size for the Dover is 5 micas).

4.6.3 Relative lengths

There are two sets of these. During galley preparation, the unit of measure is the *em*, the size of the letter M in the default font for the box. Units in this measure can be expressed as `em` or `ems`, or as `quad` or `quads`. These units should be able to be used in both horizontal and vertical directions, but it's here that I made a mistake, and you can only use them in the horizontal direction at present. During page layout, the relevant measure is `pagewidth`, the width of the page for the target device.

In the vertical direction, the appropriate measures are `line` and `lines`, and `pageheight` respectively. The size of a line is considered equal to the height of the bounding box for the default font — not an admirable choice, but at least precise.

4.6.4 Modifying environment parameters

In addition to setting environment parameters using the appropriate units, the current value may be modified. For example,

```
@begin(multiple, width -1in, above *2, below /2, need +3cms)
```

will decrease the value of the `width` parameter by one inch, double the `above` parameter, halve the `below` parameter, and increase the `need` parameter by three centimeters.

4.7 The standard values for the environment values

Below are the tables of the standard values for the environment parameters that are associated with the environment when the `box` routine is invoked. Environment parameters typically are derived in three ways — they are inherent properties of the environment (for example, a centred environment has `JustifyLeft`, `JustifyRight`, `JustifyLeftLast`, `JustifyRightLast` all `False`); they are inherited from the parent's environment parameters (for example the `width` is normally inherited in this way); and they are inherited from the extra style parameters (for example, the `Above`, `Below` and `Gap` are normally inherited from the style parameters). In addition to these three ways, the parameters may be set using parameter modifiers specified by the `Define` or `Modify` commands, or specified at the point of invoking the environment.

The interpreter associated with the environment is one of `Text`, `DP`, `Plot` or `Maths`. The computation values are listed in the order `CompLM`, `CompRM`, `CompGap`, `CompXPosn`, `CompYPosn`, `CompWidth`, `CompFont`. The width of the box is abbreviated to `W`; one eighth of the width is abbreviated to `W8`. If explicit constant values are given, the environment sets the parameters to these values; otherwise parameters are inherited from the parent environment. Justifications inherited from the parent are written as `JL`, `JR`, `JLL`, `JRL`; a face code inherited from the parent is written as `FC`, and a font size as `FS`; a border style inherited

from a parent is written as BStyle. Be warned that these tables do not yet show all the inheritances of the environments.

	EnvCentre0	EnvDisplay0	EnvExample0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	NAbove, NBelow	NAbove, NBelow	NAbove, NBelow
Margins	RLM, RRM	NLM, NRM	NLM, NRM
Continue	Disallowed	Disallowed	Disallowed
Gap	NGap	NGap	NGap
Justifications	F, F, F, F	T, F, T, F	T, F, T, F
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Normal, Roman	Normal, Roman	Normal, Typewriter
Underlining	Off	Off	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	RAbove*2	RAbove*2	Box Y size
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	7	7	7

	EnvProgramEx0	EnvEquation0	EnvFlushLeft0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	NAbove, NBelow	NAbove, NBelow	NAbove, NBelow
Margins	NLM, NRM	NLM, NRM	RLM, RRM
Continue	Disallowed	Disallowed	Disallowed
Gap	NGap	NGap	NGap
Justifications	T, F, T, F	T, F, T, F	T, F, T, F
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Normal, Algol	Normal, Typewriter	Normal, Roman
Underlining	Off	Off	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	Box Y size	Box Y size	RAbove*2
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	7	7	7

	EnvFormat0	EnvPageCommand0	EnvVerbatim0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	NAbove, NBelow	NAbove, NBelow	NAbove, NBelow
Margins	RLM, RRM	RLM, RRM	RLM, RRM
Continue	Disallowed	Disallowed	Disallowed
Gap	NGap	NGap	NGap
Justifications	T, F, T, F	T, F, T, F	T, F, T, F
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Normal, Roman	Normal, Roman	Normal, Typewriter
Underlining	Off	Off	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	RAbove*2	RAbove*2	RAbove*2
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	7	7	7

	EnvAlign0	EnvFlushRight0	EnvQuotation0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	NAbove, NBelow	NAbove, NBelow	NAbove, NBelow
Margins	RLM, RRM	RLM, RRM	NLM, NRM
Continue	Disallowed	Disallowed	Allowed
Gap	NGap	NGap	NGap
Justifications	T, F, T, F	F, T, F, T	Justifications
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Normal, Typewriter	Normal, Roman	Normal, Roman
Underlining	Off	Off	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	RAbove*2	RAbove*2	RAbove*2
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	1, 0, 0, 0, 0, 0
Tabulations	7	7	7

	EnvDefault0	EnvVerse0	EnvMajorHeading0
Interpreter	Text	Text	InterpreterText
Width	Width	Width	Width
Above and Below	RAbove, RBelow	NAbove, NBelow	RA+4*RG, RB+2*RG
Margins	RLM, RRM	RLM, RRM	RLM, RRM
Continue	Allowed	Disallowed	Disallowed
Gap	RGap	NGap	RGap+2*HGap
Justifications	Justifications	T, F, T, F	F, F, F, F
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Normal, Roman	Normal, Roman	Extra large, Bold
Underlining	Off	Off	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	RAbove*2	RAbove*2	RAbove*5
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	1, 0, 0, 0, 0, 0	2, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	7	7	7

	EnvHeading0	EnvSubHeading0	EnvChapter0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	RA+3*RG, RB+RG	RA+RG, RBelow	RA+4*RG, RB+2*RG
Margins	RLM, RRM	RLM, RRM	RLM, RRM
Continue	Disallowed	Disallowed	Disallowed
Gap	RGap+HGap	RGap-HGap	RGap+2*HGap
Justifications	F, F, F, F	T, F, T, F	F, F, F, F
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Large, Bold	Normal, Bold	Extra large, Bold
Underlining	Off	Off	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	DefaultPS
Need	RAbove*4	RAbove*3	RAbove*5
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	7	7	7

	EnvPrefaceSec0	EnvAppendix0	EnvSection0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	RA+4*RG, RB+2*RG	RA+4*RG, RB+2*RG	RA+4*RG, RB+2*RG
Margins	RLM, RRM	RLM, RRM	RLM, RRM
Continue	Disallowed	Disallowed	Disallowed
Gap	RGap+2*HGap	RGap+2*HGap	RGap+2*HGap
Justifications	F, F, F, F	F, F, F, F	T, F, T, F
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Extra large, Bold	Extra large, Bold	Extra large, Bold
Underlining	Off	Off	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	DefaultPS	DefaultPS	Skip
Need	RAbove*5	RAbove*5	RAbove*5
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	3, 0, 0, 0, 0, 0
Tabulations	7	7	7

	EnvAppendixSec0	EnvAppendix1	EnvSubSection0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	RA+4*RG, RB+2*RG	RA+4*RG, RB+2*RG	RA+3*RG, RB+RG
Margins	RLM, RRM	RLM, RRM	RLM, RRM
Continue	Disallowed	Disallowed	Disallowed
Gap	RGap+2*HGap	RGap+2*HGap	RGap+HGap
Justifications	T, F, T, F	T, F, T, F	T, F, T, F
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Extra large, Bold	Extra large, Bold	Large, Bold
Underlining	Off	Off	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	RAbove*5	RAbove*5	RAbove*4
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	3, 0, 0, 0, 0, 0	3, 0, 0, 0, 0, 0	3, 0, 0, 0, 0, 0
Tabulations	7	7	7

	EnvAppendixSec1	EnvParagraph0	EnvItemize0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	RA+3*RG, RB+RGap	RA+RGap, RBelow	RAbove, RBelow
Margins	RLM, RRM	RLM, RRM	NLM+W8/3.5, NRM
Continue	Disallowed	Disallowed	Disallowed
Gap	RGap+HGap	RGap-HGap	0
Justifications	T, F, T, F	T, F, T, F	JL, JR, JLL, JRL
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Large, Bold	Normal, Bold	Normal, Roman
Underlining	Off	Off	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	RAbove*4	RAbove*3	RAbove*2
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	3, 0, 0, 0, 0, 0	3, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	7	7	0

	EnvEnumerate0	EnvItem0	EnvDescription0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	RAbove, RBelow	NAbove, NBelow	NAbove, NBelow
Margins	NLM+W8/3.5, NRM	0, 0	0, 0
Continue	Disallowed	Disallowed	Disallowed
Gap	0	NGap	NGap
Justifications	JL, JR, JLL, JRL	JL, JR, JLL, JRL	JL, JR, JLL, JRL
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Normal, Roman	Font size, face code	Normal, Roman
Underlining	Off	UnderLine	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	RAbove*2	Need	RAbove*2
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	0	7	6

	EnvDItem0	EnvMultiple0	EnvCommentary0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	NAbove, NBelow	RAbove, RBelow	RAbove, RBelow
Margins	RLM, RRM	0, 0	RLM + W8/4, RRM + W8/4
Continue	Disallowed	EnvContinue	Disallowed
Gap	NGap	NGap	NGap
Justifications	JL, JR, JLL, JRL	JL, JR, JLL, JRL	JL, JR, JLL, JRL
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Font size, Face code	FS, FC	Normal, Roman
Underlining	UnderLine	UnderLine	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	Need	Need	Box Y size
Border size	0	Border	0
Border style	NoBorder	BStyle	NoBorder
Computations	4, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	Tabs	Tabs	7

	EnvTextPart0	EnvGloss0	EnvPlot0
Interpreter	Text	Text	Plot
Width	Wd * 3/5	W - W(siblings)	Width
Above and Below	NAbove, NBelow	NAbove, NBelow	RAbove, RBelow
Margins	RLM, RRM + W8/8	RLM + W8/8, RRM	RLM, RRM
Continue	Disallowed	Disallowed	Disallowed
Gap	NGap	NGap	NGap
Justifications	JL, JR, JLL, JRL	JL, JR, JLL, JRL	JL, JR, JLL, JRL
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	FS, FC	Succ (FS), FC	Normal, Roman
Underlining	UnderLine	UnderLine	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	Need	Need	Box Y size
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	3	3	None

	EnvDP0	EnvFigure0	EnvCaption0
Interpreter	DP	Text	Text
Width	Width	Wd * 3/5	Width
Above and Below	RAbove, RBelow	RAbove, RBelow	NABove, NBelow
Margins	RLM, RRM	RLM, RRM	RLM, RRM
Continue	Disallowed	Disallowed	Disallowed
Gap	RGap	NGap	NGap
Justifications	JL, JR, JLL, JRL	JL, JR, JLL, JRL	JL, JR, JLL, JRL
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Normal, Roman	FS, FC	FS, FC
Underlining	Off	UnderLine	UnderLine
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	Box Y size	Box Y size	Need
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	None	3	3

	EnvAnnotation0	EnvFoot0	EnvPageHeading0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	NABove, NBelow	NABove, NBelow	Above, Below, 0, 0
Margins	RLM, RRM	RLM, RRM	RLM, RRM
Continue	Disallowed	Disallowed	Disallowed
Gap	NGap	NGap	NGap
Justifications	JL, JR, JLL, JRL	JL, JR, JLL, JRL	JL, JR, JLL, JRL
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Normal, Roman	Normal, Roman	Normal, Roman
Underlining	Off	Off	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	RAbove*2	RAbove*2	0
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	None	None	None

	EnvPageFooting0	EnvPageOffset0	EnvResearchCr0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	0, 0	Above, Below, 0, 0	RA, RBelow
Margins	RLM, RRM	RLM, RRM	RLM, RRM
Continue	Disallowed	Disallowed	Disallowed
Gap	NGap	NGap	NGap
Justifications	JL, JR, JLL, JRL	JL, JR, JLL, JRL	JL, JR, JLL, JRL
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Normal, Roman	Normal, Roman	Normal, Roman
Underlining	Off	Off	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	0	0	Box Y size
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	None	None	7

	EnvAbstract0	EnvNotice0	EnvCopyrightNO
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	RAbove, RBelow	RA, RBelow	RAbove, RBelow
Margins	RLM, RRM	RLM, RRM	RLM, RRM
Continue	Disallowed	Disallowed	Disallowed
Gap	NGap	NGap	NGap
Justifications	JL, JR, JLL, JRL	JL, JR, JLL, JRL	F, F, F, F
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Normal, Roman	Normal, Roman	Normal, Roman
Underlining	Off	Off	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	Box Y size	Box Y size	Box Y size
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	7	7	7

	EnvTitleBox0	EnvTitlePage0	XItem0
Interpreter	Text	Text	Text
Width	Width	Width	W - LM - RM
Above and Below	RAbove, RBelow	RAbove, RBelow	NAbove, NBelow
Margins	RLM, RRM	RLM, RRM	0, 0
Continue	Disallowed	Disallowed	Disallowed
Gap	RGap	RGap	NGap
Justifications	JL, JR, JLL, JRL	JL, JR, JLL, JRL	JL, JR, JLL, JRL
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Normal, Roman	FS, FC	FS, FC
Underlining	Off	UnderLine	UnderLine
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	TitlePS	Skip
Need	0	0	Need
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	None	none	None

	XLabel0	XLMargin0	XRMargin0
Interpreter	Text	Text	Text
Width	LM	WB	WB
Above and Below	0, 0	RAbove, RBelow	RAbove, RBelow
Margins	RLM, 0	RLM, RRM	RLM, RRM
Continue	Disallowed	Disallowed	Disallowed
Gap	0	RGap	RGap
Justifications	T, F, T, F	JL, JR, JLL, JRL	JL, JR, JLL, JRL
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	FS, FC	FS, FC	FS, FC
Underlining	UnderLine	UnderLine	UnderLine
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	0	0	0
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	None	None	None

	EnvText0	EnvReport0	EnvArticle0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	RAbove, RBelow	RA, RBelow	RA, RBelow
Margins	RLM, RRM	RLM, RRM	RLM, RRM
Continue	Disallowed	Disallowed	Disallowed
Gap	RGap	RGap	RGap
Justifications	JL, JR, JLL, JRL	JL, JR, JLL, JRL	JL, JR, JLL, JRL
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	-	-	-
Underlining	UnderLine	UnderLine	UnderLine
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	0	0	0
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	None	None	None

	EnvThesis0	EnvSlides0	EnvManual0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	RAbove, RBelow	RAbove, RBelow	RA, RBelow
Margins	RLM, RRM	RLM, RRM	RLM, RRM
Continue	Disallowed	Disallowed	Disallowed
Gap	RGap	RGap	RGap
Justifications	JL, JR, JLL, JRL	JL, JR, JLL, JRL	JL, JR, JLL, JRL
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	-	-	-
Underlining	UnderLine	UnderLine	UnderLine
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	0	0	0
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	None	None	None

	EnvMFoot0	EnvMAnnotation0	EnvMOddsandSods0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	RAbove, RBelow	RAbove, RBelow	RAbove, RBelow
Margins	RLM, RRM	RLM, RRM	RLM, RRM
Continue	Disallowed	Disallowed	Disallowed
Gap	RGap	RGap	RGap
Justifications	JL, JR, JLL, JRL	JL, JR, JLL, JRL	JL, JR, JLL, JRL
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	-	-	-
Underlining	UnderLine	UnderLine	UnderLine
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	0	0	0
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	None	None	None

4.8 Slug Environments

Slug environments define the appearance of the items within a box. For example, they specify the font size and face codes to be used for characters, whether underlining is to be used, etc. Slug environments may be nested, for example `@b(@+(3))`. A slug environment must finish in the same box as it starts.

For the purpose of description, it is convenient to classify the slug environments, as follows.

4.8.1 Face Codes

The following are available:

<code>@r[roman]</code>	The roman (non-italic, non-bold) face.
<code>@i[italic]</code>	The <i>italic</i> face.
<code>@b[bold]</code>	The bold face.
<code>@c[Small Caps]</code>	The SMALL CAPS face. Note that upper-case produces LARGE CAPITALS, lower case SMALL CAPITALS. To see how this effect is achieved, see section 4.9.4.
<code>@g[greek]</code>	The $\gamma\rho\epsilon\epsilon\kappa$ type face.
<code>@t[typewriter]</code>	The typewriter face.
<code>@p[bold italic]</code>	The <i>bold italic</i> face.
<code>@z[symbols]</code>	A collection of mathematical $\sigma\psi\mu\beta^{\circ}\lambda\sigma$.
<code>@m0[maths font 0]</code>	The maths font 0 used in the maths environment.
<code>@m1[maths font 1]</code>	The <i>maths font 1</i> used in the maths environment.
<code>@m2[maths font 2]</code>	The $\int\leftarrow\{\nabla\Delta\leftarrow\in$ used in the maths environment.

Using a face code slug environment causes the text within it to use the specified face code in the current size of font. Note that the a font is no longer available, and that the z font is only retained for compatibility with old .Mss files: the maths fonts now provide the facilities of the z font.

4.8.2 Font Sizes

The following font sizes are available:

<code>@11[extra large]</code>	An extra large font.
<code>@1[large]</code>	A large font.
<code>@n[normal]</code>	The normal font size.
<code>@s[small]</code>	A small font.

@ss[extra small] An extra small font.

The font sizes are not absolute — in a galley that specifies its default font size to be 8 point (for example), then 1 may specify a font of point size 11, which could be the normal font size for some other environment.

If some combination of font size and face code has not been defined for the galley (see section 4.3.3), then Mint will supply some default font.

4.8.3 User Face Codes

In addition to the face codes above, which usually have a fixed meaning in each environment, there are ten other face codes (in each of the font sizes) that are free for the user to bind to specific fonts as he wishes. These face codes are f0, f1, f2, ..., f9. Section 4.9.1 describes how to associate specific fonts with these face codes.

4.8.4 Underlines, Overlines and Eraselines

The following are available:

@u[phrase]	<u>Underlines non-blank characters.</u>
@ux[phrase]	<u>Underlines all characters.</u>
@un[phrase]	<u>Underlines alphanumeric (letters & digits) only.</u>
@o[phrase]	<u>Overlines non-blank characters.</u>
@ox[phrase]	<u>Overlines all characters.</u>
@on[phrase]	<u>Overlines alphanumeric (letters & digits) only.</u>
@e[phrase]	Eraselines non-blank characters.
@ex[phrase]	Eraselines all characters.
@en[phrase]	Eraselines alphanumeric (letters & digits) only.

Several non-conflicting combinations can be used (like this).

4.8.5 Raster Functions

The raster function used to display information within a box determines how the pixels of the new information are combined with the pixels of the information that is already in the box at the position the information is being placed. The implied operation is

`Destination := Destination RasterOp Source`

where `RasterOp` is a bitwise operation. The facility is device dependent: each device specifies in its device characteristics (see section 4.13) which `RasterOps` are available. The following are available in Mint.

<code>RRp1</code>	Replace Destination by Source.
<code>RNot</code>	Replace Destination by the inverted pixels of the Source.
<code>RAnd</code>	Replace the Destination by the conjunction of the Destination and the Source.
<code>RAndNot</code>	Replace the Destination by the conjunction of the Destination and the inverted Source.
<code>ROr</code>	Replace the Destination by the disjunction of the Destination and the Source.
<code>ROrNot</code>	Replace the Destination by the disjunction of the Destination and the inverted pixels of the Source.
<code>RXOr</code>	Replace the Destination by the exclusive <code>or</code> of the Destination and the Source.
<code>RXNor</code>	Replace the Destination by the exclusive <code>nor</code> of the Destination and the Source.

All these raster functions are available on the Perq; only `ROr` is available on the Dover. All the environments have `ROr` as their default `RasterOp`.

4.8.6 Scripting

The following scripting slug environments are available:

<code>@+(phrase)</code>	Super ^{scripts} the phrase.
<code>@-(phrase)</code>	Sub _{scripts} the phrase.

The amount of the baseline shift is determined by the font size of the text, and is not currently able to be altered by the Minter. In general you should use the `maths` environment for superscripting and subscripting. See section 4.27.

4.8.7 Overprinting

The `@ovp` environment remembers the current position in the slug, and then backs the slug up to that point again when the environment finishes. `@Ovp` may be used for hacking new symbols, such as †, or for primitive mathematics, such as $X\frac{1}{2}$. `Ovp` environments can be nested.

4.9 Fonts

Mint can handle several different font representations (simultaneously if needed — this occurs when cross-proofing, or if a document is being sent to several galleys, each with a different associated device). Fonts are accessed by a galley by indexing into the rectangular array of *Font Indicators* associated with each galley. The rectangular array is indexed by *Font Size* and *Face Code*; see sections 4.8.1 and 4.8.2 for more information about these terms. A font indicator must be written into the appropriate position in the array for the galley to be able to use the given font size and face code. Some font indicators are written into the font arrays when the standard galleys are created; see section 4.4.2 for which ones are put there. If a position is accessed which does not have a font indicator, Mint will supply a (device specific) default font.

It is possible to associate font indicators with the elements of the array, and to overwrite the ones that are loaded there initially. This section describes how to do this, and how to tailor fonts by selectively replacing some characters by characters from other fonts.

4.9.1 Associating Fonts with a galley

A font is associated with some element of the font array for some galley by using the statement `AssocFont`. This takes a galley identifier, the element position, and the name for the font. Mint can accept both Xerox and T_EX font names.

```
@assocfont(main, n, z, zfont10)
```

`AssocFont` does not cause the font to be loaded (or the fonts width information to be loaded). Only when information from the font is needed is a check made to see if the font information is already available, and if not, it is loaded into memory. Note that Mint uses the device that has been associated with the font to determine the location of the font information, and understand its representation. If the document is being prepared for several devices simultaneously, Mint will use the device identifier to disambiguate the font identifiers (just as you would expect).

4.9.2 Modifying fonts

It is frequently desirable to substitute a character in one font by a character from another font. This saves frequent changes of slug environment, for example. In general, though, merely replacing one character by another is not sufficient. You may want to introduce into your document an *icon*, which is some picture created by (say) a drawing package, and have the picture displayed instead of some character. In this way you can build up new characters that are not found in the fonts available on the target device, without having to go to the labour of creating a whole new font (using, for example, MetaFont).

Mint provides a general mechanism for replacing a character in one font by a character in another, for replacing a character by a blankspace of some user-specified size, and for replacing a character by an icon.

The only restrictions are that the fonts and icons have all been created for the same device (naturally); the substitution process is transitive, so that it is possible to substitute a character which has itself already been substituted.

Four facilities are provided.

- A single character can be substituted by the statement

```
@substitutechar(dover,timesroman10r,@char(#17),symbol12,@char(#12))
```

which replaces character #17 in Dover font TimesRoman10r by character #12 from font Symbol12 (a fairly eccentric thing to do, but there's no accounting for lack of taste).

- A range of characters can be substituted using `SubstituteRange` which takes a range of characters in the destination font, and a starting character in the source font. It acts in the same way as repeated use of `SubstituteChar`. For example,

```
@substituterange(Dover, TimesRoman10i, A, Z, TimesRoman10b, a)
```

will cause all upper case letters in TimesRoman10i to appear as lowercase bold letters.

- A character can be replaced by a gap of some specified size by `SubstituteGap`. This will cause the output of a gap of the specified size instead of the character. This feature is of use for creating narrow gaps, for italic corrections, for example.

```
@SubstituteGap (Perq, Gacha9, @char(sp), 5 rasters)
```

The gap can be specified in rasters, ems or quads, or absolute units.

- A character can be replaced by an icon by the statement `SubstituteIcon`. Icons are dealt with in the next section.

4.9.3 Icons

One way of looking at a font is to regard it as a collection of representations of characters, which are displayed by an interpreter (let's call it the *text interpreter*) when the character is to be displayed. Usually the same interpreter is invoked for all the characters of the font. We are not usually concerned with the way the interpreter works, or how the representations of the characters are stored. For example, the characters may be stored as bit-maps, or as splines, and the interpreter may determine where in the page the pixels should be placed; alternatively, the characters may be stored as raised pieces of metal on a daisy wheel, and the interpreter works by selecting the appropriate leaf, and bashing it against the paper.

The interpreters above have the luxury of being able to work the same way for all characters. Life in the real world is seldom so simple, though. Sometimes it is not possible to find the *glyph* that you would like in some font; for example, you might want to use a small picture of the mouse buttons, with one of the buttons blackened, to allow you to say "press button " instead of "press the yellow button on the mouse". Mint allows you to associate an arbitrary glyph with a character in a font, using the `MakeIcon` and `SubstituteIcon` statements. These allow you to invoke an arbitrary interpreter to display information in the document; currently you will have to read "DP" for "arbitrary" in this sentence, but that will change shortly.

An icon must first be prepared using `DP` — since the icon will normally be greatly reduced in size when it is printed, the icon should be made very simple. An icon is then associated with an identifier with the `MakeIcon` statement

```
@MakeIcon (DP, Mouse, SimpleMouse.DP)
```

which associates the `DP` drawing in file `SimpleMouse.DP` with the identifier `Mouse`. Mint will store the representation of the drawing, with information about the aspect ratio. To substitute the icon for some character, Mint requires several pieces of information — the size that the icon will be drawn; the `X` and `Y` positions of the top left-hand corner of the bounding box, relative to the current position in the slug that is being made; and the width of the icon. These four pieces of information allow you to adjust the icon horizontally and vertically relative to the other characters on the line. Note, however, that it does not cause the line to be thicker in the vertical direction, because Mint assumes that the nominal height is the same as the height of the font in which the icon has been substituted.

To substitute an icon, use

```
@substituteicon (Dover, TimesRoman10, 0, Mouse, 0.16in, 0in, 0.20in, 0.12in)
```

which specifies that `0` in `TimesRoman10` on the `Dover` will be replaced by the mouse icon, scaled to `0.16in`, with `X` and `Y` positions `0in` and `0.20in`, and the nominal width `0.12in`. I leave it as a simple exercise to work out how the tail of the mouse has been drawn.



4.9.4 New fonts

The statements `SubstituteChar`, `SubstituteRange`, `SubstituteGap` and `SubstituteIcon` cause changes to the copy of the font stored internally. Since normally only one copy of the font information is kept, and a font can occur at several positions in the font arrays of the galley, these statements can cause surprising side-effects. The well-disciplined hacker will take a private copy of any font that he is about to modify; this is done using `CopyFont`.

```
@CopyFont (Dover, MyDirtyFont, Saila10)
```

In this case the font substitutions should be made on `MyDirtyFont`, which can then be associated with some element in the font array. Any substitutions that have been made to `Saila10` before the `copyfont` statement will be copied into `mydirtyfont`.

The `c` slug environment has been created using

```
@copyfont(Dover, CapsFont, TimesRoman10)  
@substituterange(Dover, CapsFont, a, z, TimesRoman8, A)  
@assocfont(Main, n, c, CapsFont)
```

Sometimes it is not meaningful to start with some existing font when doing the `char`, `gap` and `icon` substitutions; you simply want to start off with an empty font, and place all the characters, gaps and icons there yourself. The `EmptyFont` statement creates a new, empty font. This font has to be specified to be used with some device, and has to have its height and baseline given, using the `Dover` font conventions. After it has been created, characters can be filled in using the `substitute` statements.

The statement

```
@EmptyFont (Dover, MyFont, 12points, -1point)
```

creates a new empty font, of height 12 points, with the baseline one point below the origin.

4.9.5 The mathematics fonts

The mathematical fonts are treated in more detail in section 4.27.4.1; here I will just describe the statements for manipulating the maths extension font, which contains several characters useful for doing high-quality mathematical typesetting.

The statement `createmfont` creates a new maths extension font from some standard (Xerox or `TEX`) font. The object that is created has additional information that is needed for mathematical typesetting.

```
@CreateMFont (Device=Dover, DFont=MintMexFont, SFont=CMathX10S10)
```

This font can now be associated with galley's using `assocmfont`:

```
@AssocMFont (Galley=Main, FontName=MintMexFont)
```

when it becomes available for use in the `maths` environment.

Sometimes the information associated with a character in the maths extension font is not suitable or accurate enough to do refined maths typesetting; if this is the case, the information can be altered using `setmexchar`. For example

```
@SetMexChar (Device = Dover, DFont = MintMexFont, DChar = @Char(#130),
              Y0 = -493 micas, YY = 0 micas, YRel = 357 micas)
```

alters the information for the large sigma character, to lift it slightly above the position it occupies normally, in order to avoid a slight bottom-heaviness that mars its use in `TEX`. Normally you will not be concerned with these refinements, since they are all available in the `Maths.Fnt include file`.

4.9.6 Character information

Sometimes it is the case that you need information about a character, such as its width, to feed into another statement. For example, you might want to find the width of a space in one font, so as to place a gap

of that size in some other font. The statements `charinfo` and `mexcharinfo` extract the information as a string, so that they can be used as arguments to other statements.

The form of these statements is

```
@CharInfo (Dev = Dover, Font = TimesRoman10, C = @Char(Sp), Info = XSize)
@MexCharInfo (Dev = Dover, Font = MintMexFont, C = @Char(Sp), Info = XSize)
```

which will extract the information in the form `l8raster`. The `info` parameter takes the values `x0`, `y0`, `xx`, `yy`, `xsize` and `ysize`. `x0` and `y0` measure the distance of the bottom left-hand corner from the origin of the character; `xx` and `yy` measure the distance of the top right-hand corner from the origin of the character; `xsize` and `ysize` are the sizes of the bounding box of the character.

4.10 Colours

Although the majority of devices allow only two colours — black and white — Mint provides a general mechanism for producing coloured output; whether these facilities will be useable on any particular device will depend upon the device's characteristics, and upon whether the driver for the device is able to accept Press Files having colour information encoded within them. Note that the Grinnell driver takes only a subset of the facilities that Mint provides, and that anyone wanting to produce coloured output (e.g. for slides) should consult me before doing so.

Mint's facilities work by providing a means of overlaying objects of one colour by objects of another colour. Objects are either completely colourless and transparent, so that objects beneath them show through without any change in the colour; or they are completely opaque and coloured, and obscure totally the objects beneath them. To determine the appearance of a document requires, therefore, a knowledge of the order in which objects are laid down.

In this section I first describe how to specify colours, and how to specify how to colour objects, and then describe the order of overlaying.

4.10.1 Defining colours

Mint uses a widely accepted colour encoding known as the *Munsell colour encoding*. In this encoding a colour is described by three values: the colour's hue, which specifies ...; the colour's saturation, which specifies its ...; and the colour's brightness, which specifies its In terms of these parameters, white has the values $(0, 0, 255)$ and black has the values $(0, 0, 0)$. Full saturated red has the values (p, q, r) ; full saturated green has the values (p, q, r) ; and full saturated blue has the values (p, q, r) .

Within Mint an identifier is associated with every colour. To associate a colour with an identifier use the statement

```
@NewColour (BrightRed, x, y, z)
```

This specifies that the identifier `BrightRed` will have the corresponding hue, saturation and brightness (in that order). If the same identifier is used more than once, the most recent definition is used.

Three colours are predefined within Mint. They correspond to the following definitions.

```
@NewColour (Black, 0, 0, 0)
@NewColour (White, 0, 0, 255)
@NewColour (GreyHT, 0, 0, 155)
```

`GreyHT` is a grey that is recognised by the Dover driver; an object that is coloured `greyht` will be shaded using the `grey dover` font. The effect is not too satisfactory, because the object has to be filled in by a mosaic of characters which may not totally fill the area being coloured, and because the xerographic copying technology is very bad at colouring areas.

In addition to the colours above, a special colour, `transparent`, is defined. Objects that are specified to be `transparent` will allow the underlying objects to show through (though *images* are not allowed to be transparent; see below).

4.10.2 Associating colours with objects

Colours can be associated with page areas, with box backgrounds, with box and area borderstyles, and with lines and characters drawn within boxes. Information about how to specify colours for these objects is given in the relevant sections; it is collected here for convenience.

4.10.2.1 Associating colours with page areas

A colour can be associated with each page area. `Transparent` is associated by default; the colour is changed by the statement

```
@PageArea (Default, Main, Red, Width1)
```

which specifies that all pages made by the `default` page layout routines will have their `main area` coloured `red`, and have a border that is specified by `width1`. The layout routines that can be used in this way are `default`, `titlepage` and `pasteup`; the corresponding page areas are given in section **??**.

4.10.2.2 Associating colours with boxes

The box environment parameter `backgroundcolour` sets the colour of the background. For example

```
@modify(Figure, BackgroundColour = Yellow)
```

will set the background colour of all figures to be whatever the colour `yellow` has been defined to be. The background colour is inherited in the same way as other box environment parameters; so that after the statement above, the statements

```
@begin(figure)
  @dp(@include(mouse.dp))
  @caption(A cowardly mouse)
@end(figure)
```

will cause both the DP drawing and the caption to have yellow backgrounds.

4.10.2.3 Associating colours with borders

Each of the lines of a pattern that is created using the statement `newpattern` can be drawn a different colour, allowing multi-coloured borders to be created. For example

```
@NewPattern (Patriotic, 8, 1, Red, 8, 1, White, 8, 1, Blue)
```

will draw a border pattern that has a line that is $8/100^{\text{th}}$ inch red, $8/100^{\text{th}}$ inch white, and $8/100^{\text{th}}$ inch blue. The default colour is black, and the colour specified for a line style of 0 is ignored, allowing the background colour to show through.

4.10.2.4 Associating colours with characters and lines

Objects that are drawn within boxes can be coloured in two ways: First, the box environment parameter `imagecolour` can be used to determine the colour of all the objects drawn in the box (for example, characters from any font, lines drawn by DP, and by Plot); second, colours can be used in the same way as slug environments to cause local colour changes. Images cannot be specified to be `transparent`.

For example

```
@begin(caption, imagecolour=green)
```

will cause all the colours in the caption to be coloured green, and (since `imagecolour` is inherited in the usual way)

```
@make (slides, imagecolour blue)
```

will cause all the object drawn in boxes (and in particular, the lines drawn by DP and Plot, and the characters in all the other boxes), to be coloured blue. The `imagecolour` is inherited in the usual way; if it is not otherwise specified in the style parameters it is set to `black`.

If you just want to colour a few characters a different colour from the current `imagecolour`, you can use any of the colours that you have defined in the same way as a slug environment. For example

```
@begin(description, imagecolour brown)
@red(Red)@ \is used for the polysilicon layer
```

```
@blue(Blue)@\\is used for the metal conductors
@end(description)
```

will cause the word `Red` to be coloured `red`, the word `Blue` to be coloured `blue`, and the rest of the letters to be coloured `brown`. (If you define a colour to use the same identifier as a slug environment, the slug environment will take precedence.)

4.10.3 The order of overlaying

All objects are completely opaque, unless they have been specified to be `transparent`, in which case they allow the colour of the objects beneath them to show through without modification.

The order in which objects are laid down is as follows: First, each page area is laid down with its border, in the order shown in section 4.16.2.3; next, each box is laid down with its border, in the inverse order of nesting (thus a `figure` box is laid down before a `plot` box and a `caption` box); finally, the images within the box are laid down. I'm not willing to specify the order that images get laid down in the case where one image overlays another one.

4.11 Computations

During the processing of the text in a galley, several pieces of information are required by the slug layout routine in order for it to be able to format the slugs and boxes. For example, the sizes of the margins, the fonts to use, and the sizes of the boxes are needed. Usually, for technical documents, the information can be presented to the slug layout routine in a straightforward way — the margins are generally fixed, the same font is used throughout, and the size of the box is simply determined from the size of its parent. The complexity that is needed for advertising copy, where baselines of slugs may not be straight, where the size of characters may change along the line, and where text may wrap around diagrams, is not needed for technical documents. However, occasionally you do need more control over these parameters than is usually supplied.

Mint gives you this control, though in a fairly crude way since I am not certain yet what is needed. Each of the parameters required by the layout routine is obtained by calling a *computation*, a Pascal procedure built into Mint. One of the parameters to these procedures is the *computation number* provided through the environment parameters of the box in which the slug will be placed. The computations that are built in handle the standard cases, for example the crooked left margin for the `description` environment; further built-in routines can be added as needed. The computation numbers are specified by passing small integer values to the computation parameters (`CompLM`, `CompWidth`, etc.) in the environment parameters. There is also an escape mechanism that provides the fearless document formatter the freedom to supply arbitrary computations without having to take the code of Mint apart; this mechanism is described below.

4.11.1 Standard computations

Computations are provided to set left and right margins, the gaps between the slugs, the positions of boxes relative to their neighbours, the width of boxes, and the fonts to be used in the slugs. Each in-built computation is specified by a value less than 8; values greater or equal to 8 are used to specify arbitrary computations.

CompLM	0 specifies a straight left margin; 1 specifies a margin that indents on the first line; 2 specifies a margin that indents a further distance if a line is continued into the next slug — this is used for the <code>verse</code> environment; 3 specifies a margin that indents all lines except the first, which is needed for multi-line section headings; 4 specifies a margin that indents to the first tab setting for all lines except the first, which is used for the <code>description</code> environment.
CompRM	Only one computation is provided: 0. This sets up a straight right margin.
CompGap	Only one computation is provided: 0. This sets the gap between slugs to be the value of the <code>gap</code> parameter in the environment.
CompWidth	Only one computation is provided: 0. This sets the width of the box to be the width in the environment parameters.
CompXPosn	Three computations are provided. Computation 0 centres the box within the surrounding box; computation 1 flushes the box left against the border of the surrounding box; and computation 2 flushes it right.
CompYPosn	Two computations are provided. Computation 0 computes the <i>y</i> position of the box from its <code>above</code> and the <code>below</code> of the previous box; computation 2 places the box directly under the previous box.
CompFont	Only one computation is provided: 0. This sets the font to the font in the environment parameters.

4.11.2 Arbitrary computations

In the most general case, you might want to determine the size of a character, its font, and some arbitrary transformations on it, as a function of the position of the character within the box. Doing this with reasonable efficiency is not feasible; however, I do provide a limited facility which is at a sufficiently low level that many useful (and many more useless) effects can be obtained, without reducing the efficiency of the normal text layout.

Four statements are provided. They allow arbitrary values to be yielded by the computation routines, based on the *slug number*, which is the number of the slug in the box counting from zero. These computations are associated with a *computation number*, so that several can be installed and used. The statements take the general form

```
@setcomp $xx$  (computation number, slug number, value)
```

which specifies that if you have specified `computation number`, and the slug you are processing is `slug number`, then you should pass the `value` to the layout routine.

More specifically, to set the left margin to some value, you write

```
@setcomplm (8, 0, 1in)
```

Then, if you specify `complm` to be 8 for some environment, a left margin of 1 inch will be used when creating the zeroth slug (the first to be put into the box). You should always have a computation specified for slug number -1 — this is used if no other explicit value is found. If you specify several values for some computation, the most recent value is the one that will be used.

Currently you have available `setcomplm`, `setcomprm`, `setcompgap`, which all take parameters as described above, and `setcompfont`, which takes parameters as follows

```
@setcompfont(dover, 8, 0, TimesRoman12)
```

Each of the computations operates independently, thus it is possible to achieve a variety of effects, not all of which seem to be useful, but which nevertheless illustrate the ability of Mint to handle strange and unusual type-setting situations. It should be quite easy, after this description, for even the casual Mint user to obtain the effect that I am using to lay out this paragraph. While I cannot maintain that the effect that you are seeing here is so very useful, it is the case that it has been achieved without any extra-ordinary effort. This is good news for those who have the need for unusual effects. Look out for even better effects in the future; in particular, I will be able to alter the baseline, and change the size, slope and thickness of characters along a line.

4.12 Box procedures

The box procedures form the basis of the semantic analysers associated with a galley. It is the box procedures that create the boxes, load them with slugs, and associate the boxes together in the appropriate way. They operate together as a collection of recursive routines, which call each other in a way partly determined by themselves, and partly by the document syntax. Since they are tied fairly closely to the syntax, and operate right in the guts of Mint, you should generally leave them well alone unless you are an expert. However, Mint has been designed to allow box procedures to be added by the system maintainer, so a description of them is not out of place.

A box procedure is written in a stylized form of Pascal, and is compiled into Mint. (There are facilities, currently disabled, for reading box procedures into Mint during document formatting.) In order to write a box procedure, you must understand the conventions of the stylized Pascal — this document isn't the place to these conventions⁵. They operate in conjunction with the environment indicators that get passed to them by the galley, and together they determine the layout and appearances of the boxes.

Since examining the code is not likely to yield too much information, the procedures are described informally.

- BoxStandard0** This box procedure simply reads input into slugs, and places the slugs into the boxes, until the end of the environment is reached. This procedure (like most of the others) is indifferent to which interpreter analyses the input to determine how to lay out the slug. For example it is used for most of the standard environments, and also for DP and Plot input.
- BoxSectionEnv0** This procedure is similar to **BoxStandard0**, except that it injects a prefix for the environment into the first slug that is made.
- BoxItemize0** This procedure creates two boxes that are beside each other. The left-hand one is used to contain the bullets, the right hand one is used to contain the `items`. The procedure iterates until the last item of the environment is read. The environment parameters adjust the margins appropriately so that nested `itemizes` have the correct indentations. There are two levels of bullets: solid black and open circles.
- BoxEnumerate0** This procedure is very similar to the previous one, except that it generates a sequence of labels. These nest three deep; the first level counts through the integers, the second through the letters, and the third through lower case roman numerals. Deeper nesting will cause the counter styles to repeat. You will need to take a crow-bar to Mint if you want to change the order or styles.
- BoxCRTerm0** This is a specialized procedure used by the `PageCommand` environment. It sets up a cross reference before scanning in its body in a manner similar to **BoxStandard0**.
- BoxCrossRef0** This procedure is used by all the environments that need to leave pointers from one galley into another galley. A cross reference is placed in the galley, and the routine then recursively calls some other box routine. It is used by `Foot`, `Annotate` and the page heading and footing environments.
- BoxMultiple0** This routine is another general-purpose routine, which, instead of reading in slugs, as does **BoxStandard0**, recursively invokes other box routines, as determined both by the syntax and by the input from the `.Mss` file. It is used by many of the non-terminal environments (and by `Multiple` in particular).

⁵ Because the galleys operate as independent processes, conventional Pascal is not usable. Since the amount of sharing between the processes is large, it was not appropriate to use processes provided by the operating system. The solution used by Mint is to implement a simple multi-process interpreter which executes the recursive analysers. The interpreter operates on the code that is generated from the box routines, which are *self-compiling*: when executed, they generate the code which will perform the analysis. And that is why they are written in a stylized Pascal.

<code>BoxCommentary0</code>	This was written originally to illustrate how the notion of box procedures allowed you to create environments not easily obtained in other ways. It turned out to be reasonably useful. The routine operates by invoking a sub-environment, and then checking to see if a <code>gloss</code> follows, and if so, placing it in a box adjacent to the first environment.
<code>BoxDescribe0</code>	This is a generalization of <code>BoxCommentary0</code> . It places several inner boxes side by side; the number of boxes that are placed adjacent to each other is determined by the number of tabulations within the environment passed to the procedure. The procedure can be used for a variety of display purposes — for the <code>description</code> environment and the <code>commentary</code> environment.
<code>BoxFigure0</code>	This procedure repeatedly accepts the bodies of the figures, and places a caption below each figure if there is such an environment in the <code>.Mss</code> file. Thus several figures can be collected together into one box.
<code>BoxTable0</code>	This procedure repeatedly accepts the bodies of the tables, and places a caption above each table if there is such an environment in the <code>.Mss</code> file. Thus several tables can be collected together into one box.
<code>BoxCaption0</code>	This environment is very similar to the <code>BoxSectionEnv0</code> environment, differing only because it places the prefix of its parent environment (which will be either <code>table</code> or <code>figure</code>) at the beginning of its first slug.
<code>BoxMaths0</code>	This environment is similar to the <code>BoxStandard0</code> , but it invokes some special processing to handle the <code>maths</code> environment.
<code>BoxGalley0</code>	This is the driving procedure for each of the galleys. Basically, it sits and loops over other non-terminals procedures. This is the procedure that usually gets interrupted when the inner environment ends — for example when a footnote or annotation has ended. Because the procedure loops, it is always ready to accept input, and so gets awoken whenever the galley manager provides it with lexemes.
<code>BoxText0</code>	This procedure is used to fire up the <code>text</code> document type. It differs from the procedure used by the other environments because <code>text</code> is the default environment. It calls the <code>BoxGalley0</code> box procedure.
<code>BoxDocType0</code>	This is the procedure that is invoked by all the document types except for <code>Text</code> . After ensuring that the style parameters have been appropriately dealt with, it invokes the <code>BoxGalley0</code> box procedure.
<code>BoxDocument0</code>	This is the grand-daddy of them all. It sits right at the bottom of the invocation stack of the principal galley, and causes everything else that is needed to create galleys to occur.

4.13 Devices

Mint uses information about devices to determine how it is to format a document. This information is stored in device tables, and it describes the physical and abstract properties of the device. In general, the information is inaccessible to the casual user, but since access to the information, or the possibility of altering it, may prove useful to someone, it is described here.

4.13.1 Device table information

The following information is stored in the device tables:

Device identifier	For example, <code>Perq</code> , <code>Dover</code> .
Raster Operations	The set of raster operations that the device can use. The <code>Perq</code> has all eight available, the <code>Dover</code> only <code>R0r</code> .
Page Size	The size of the page, in $1/100^{\text{th}}$ inch.
Raster Size	The size of the page in rasters (these rasters need not correspond to physical characteristics of the device). For the <code>Perq</code> the raster size is the same as the physical raster size (about $1/100^{\text{th}}$ inch); for the <code>Dover</code> the raster size is 5 micas.

From the above it should be quite clear that the device table information needs to be rethought. I'm sure it will be a refreshing experience.

4.14 Presentations

The purpose of Mint is to take a manuscript and prepare from it a number of *presentations*. A presentation comprises a collection of *document parts*, each of which comprises some number of pages. The information in the pages comes from the original manuscript, of course, but the manner in which the information is presented — the appearance of the pages, the way that cross-references are indicated, and even which portions of the original manuscript get displayed, is a property of the presentation. Some of the flexibility of having different presentations is lost because all presentations that are created during an execution of Mint come from the same set of galleys; however, the user can select which galleys will contribute to a presentation, so that this loss of flexibility may be disguised. It is possible, for example, to have two separate `Main` galleys, each receiving the document, but having different style parameters, different procedures, different fonts, and different target devices. The only restriction, in fact, is the obvious one that all the galleys that contribute to a presentation must use the same target device.

Note, however, that some of the implied richness in selecting and designing presentations is not possible with the non-interactive version of Mint, and will only be able to be exercised in the interactive version.

In this section I describe the structure of a presentation, and the means by which the default presentation can be modified.

4.14.1 The structure of a presentation

A presentation has two components: a vector that maps between *page layouts* and collections of formatting rules that lay out pages; and a collection of *document parts*, which are the pages which have been produced using the particular lay out rules. Different page layouts are specified for different portions of the document through the use of box environment parameters. For example, the `TitlePage` environment

specifies the page layout parameter of its environment to be `TitlePS`. Consequently, when a presentation comes to be made, the particular collection of rules in the `TitlePS` entry of the layout mapping vector of the presentation will be used.

Also associated with each of the possible page layouts is a part. The part receives the pages as they are created by the formatting rules, and pages can get sent to the parts in any order; within the part, though, they are placed consecutively by order of arrival. The *printing order* of the pages in a presentation is quite arbitrary, since all the pages are created before any printing is performed.

As I understand page layout requirements more, I may be able to bind more of the page layout rules as declarative information, but at present the major burden of defining the appearance of a page according to some formatting rules is specified by procedural knowledge. However it is possible to change several of the parameters the layout routines use, such as those specifying the size of the page, the style of the borders around areas, and the background colours of areas. These are dealt with in section 4.16.2.3.

4.14.2 Defining layout procedures

Well, of course, you have to write the routines that perform the page layouts, which means you need to know all the grubby details of Mint's page layout mechanisms. In order to make effective use of the parameters which the layout procedures use, it is necessary to describe the general mechanisms that specify which layouts are associated with the presentation mapping vector.

Each layout procedure has an identifier; currently Mint has three layout procedures: `TitlePage0`, which understands how to lay out title pages, `Contents0` which helps lay out tables of contents, and `Default0`, which lays out everything else. More details of these are given in section 4.16. Soon there will be layout procedures for letters, multi-column pages, etc.

4.14.3 Defining new presentations

A new presentation is created by the `NewPresentation` statement. This creates a new presentation, and sets the elements of its layout mapping vector to undefined values. Layout procedures are associated with the elements and (empty) parts created, by the `AssocPart` statement.

For example,

```
@NewPresentation (MyPres)
@AssocPart (MyPres, TitlePage, TitlePS, TitlePage0)
@AssocPart (MyPres, MainBody, DefaultPS, Default0)
```

creates a presentation named `MyPres`, and associates two parts with it. The first part, named `TitlePage`, is associated with the `TitlePS` element of the mapping vector, and specifies that the layout will be performed using `TitlePage0`. The second part, named `MainBody`, is associated with the `DefaultPS` element of the mapping vector, and specifies that the layout will be performed using `Default0`.

4.14.4 Making representations⁶

A representation of the manuscript is finally made from the galleys, by giving a presentation a collection of galleys. Each galley given to the presentation must have the same target device, but there are no other restrictions. One of the galleys in the collection is specified to be the *principal* galley; it is from this galley that slugs and boxes will be drawn initially to make the representation. The other galleys of the collection are *associated* galleys: if there are cross-reference relations from the principal galley into an associated galley, then the page layout routines will combine the boxes and slugs from the associated galley into the pages according to the routine's rules. Currently the only way this can be done is by placing the cross-referenced material into footnotes, so that the usual way of making a representation is to have the principal galley be the `Main` galley, and the associated galleys be that subset of the `Footnote` and `Annotation` galley from which it is desired to extract notations.

To make a representation, the statement `MakeRep` is used. This statement takes a principal galley, a collection of associated galleys, and a presentation:

```
@MakeRep (Main, @"(Footnote, Annotation), MyPres)
```

Note that the second parameter must be quoted. It is picked apart by the macro. (At present you cannot use this statement.)

4.14.5 Printing a presentation

After a presentation has been made, it may be printed in whole, or page by page, or part by part. There is no requirement that the printing device be the same as the target device of the galleys that went in to make the presentation. If they are not the same, they will be printed on the viewing device in cross-proofing mode; see section 4.25.

The following illustrate the methods of causing printing to occur.

```
@PrintPage (MyPres, Dover, TitlePage, 1)
```

will cause page 1 of the `TitlePage` part of presentation `MyPres` to be printed on the `Dover`,

```
@PrintRange(MyPres, Perq, MainBody, 1, 10)
```

will print pages 1 to 10, and

```
@PrintPresentation (MyPres, Dover)
```

⁶ Although this section is describing facilities that are right at the edge of the Mint design, they are sufficiently well developed that it is reasonable to program Mint at this level.

will print the whole presentation. (None of these statements can be used directly at present, though they are used implicitly when Mint interacts with you when it has formatted a document.)

4.15 Standard presentations and printing

In this section the standard presentations, parts and layout routines are described, together with the standard printing action with which Mint finishes its execution.

4.15.1 Standard presentations

The following layout routines are specified. Their actions are described in section 4.16.

```
Default0 Contents0 TitlePage0
```

The following is the standard presentation.

```
@NewPresentation (Standard)
@AssocPart (Standard, TitlePage, TitlePS, TitlePage0)
@AssocPart (Standard, Contents, ContentsPS, Contents0)
@AssocPart (Standard, MainBody, DefaultPS, Default0)
```

The table of contents is created only if there is an entry in the `ContentsPS` entry of the page layout of the presentation, and if there is a galley named `Contents`. Information is sent to the `Contents` galley automatically, and it finally comes to be laid out using the layout routine in the `DefaultPS` entry of the vector. See section **** for more details about tables of contents.

4.15.2 Printing the standard presentation

Mint creates the following representation after it has created the galleys.

```
@MakeRep (Main, FootNote, Standard)
```

After creating the presentation, Mint interacts with the user, calling `PrintPage` and `PrintPresentation`, as requested.

4.16 Layout procedures

Layout procedures are used by Mint to take the slugs and boxes from the galleys and place them into the pages. I am still a long distance from being able to formalize this activity as well as I can formalize the activity of creating the slugs and boxes in the galleys. Thus the description of the action of the page layout procedures is less precise than I desire. Nonetheless, the current page layout procedures appear to operate

fairly well, though I can imagine that they will collapse in horrible ways if presented with pathological layout problems.

In the sections below I first describe the sorting action which precedes page layout, and then describe informally the action of each of the two procedures in Mint.

4.16.1 Sorting the slugs and boxes

The galleys have a rich, detailed representation of the structure of a document; this structure is produced by a parse of the manuscript. The task of page layout is to create another structure, based on the galley structure, but which selectively ignores some of the structural information (since this is normally deduced by the reader from the representation of the document), and has imposed upon it another structure, caused by the constraints of the two-dimensional pages. One such piece of information which is hidden in the galleys but which is vital for page layout is the ordering of the slugs and boxes along the *Y*-axis. Two slugs which need to appear adjacent to each other on a page may be separated in the galley into different leaves of the structure. An example occurs with the bullets that introduce items in an itemized list — unlike Scribe, they are not in the same slug.

To assist the page layout routines a second structure is created. This is a list of all the slugs and boxes, sorted by *Y*-value. This sorted list is used to help find which slugs and boxes are intended for which layout routine. This is done by creating several collections of sorted slugs. Each collection is passed to one of the page layout procedures for processing; several consecutive collections may be passed to the same page layout procedure, but each collection will result in initializations occurring in the layout procedures that affect the appearance of the presentation (which is why each chapter reinitializes the headings and footings).

There are a few esoteric properties of the `PageStyle` environment parameters that control the general appearance of documents, but here is not the place to discuss them.

4.16.2 Page areas

Each layout procedure regards the page as being divided into a number of nested *areas*; the sizes, positions, border widths, border styles and background colours of these areas are determined by parameters that are used by the layout routines. There two sorts of parameter: parameters which set the size of the page for all the layout routines, and parameters which modify how areas appear, and which are specific to each of the layout routines.

4.16.2.1 Page parameters

The page height, page width and border width of the pages can be set. The border width is used to compute the size of the border around the text on the page, according to the tables given below (4.16.2.3).

The following statement sets the page parameters.

```
@PageParams (PageHt = 10 inches, PageWd = 8 inches, BorderWd = 1 inch)
```

All the parameters must be set, and they must be in absolute units.

If the width of the main text area is changed (the area into which the slugs are placed), then the width of the galley should also be changed to be the same, otherwise page layout will give unpredictable results.

4.16.2.2 Area parameters

It is possible to change the border width, border style, and background colour of each of the areas for each of the layout routines. The parameters affect all the pages that are made by the routine. To alter the parameters, use, for example

```
@PageArea (Layout = TitlePage, Area = Main, BackgroundColour = Red,
           Border = 0.05in, BorderStyle = Width1)
```

This will draw a `width1` border round the main area of all the pages made by the `titlepage` layout routines (there's only one at present: `titlepage0`), leave a border of width 0.05 inches, and colour the background of the area `red`. The `layout` parameter can be `default` or `titlepage`; the `area` parameter should be the identifier of one of the areas in the pages used by that layout routine. The `layout` and `area` parameters cannot be omitted; other parameters that you don't want to change can be omitted.

If you put a border round the main area, then you will need to alter the width of the galleys, since the slugs are placed into the main area within the inner borders of the area.

4.16.2.3 Values of the page area parameters

The following are the areas specified by the current layout routines. The parameters are presented in the order: *parent area*, within which this area is nested; the *sibling area*, which the next area with the same parent; the *son area*, which is one of the areas into which the considered area is divided; and then four distances: the coordinates of the top left-hand corner relative to the parent, and the *x* and *y* sizes of the area.

The values of `PH`, `PW` and `BW` are set by the `pageparams` statement; their default values are the page height and width, as specified by the device characteristics, and one eighth of the page width, respectively. The border widths, styles and background colours are set by the `pagearea` statement; their default values are zero, `noborder` and `transparent`, respectively.

Default

Page						
NoArea	NoArea	Heading	0	0	PW	PH
Heading						
Page	Body	LeftMargin	0	0	PW	BW
Body						
Page	Footing	NoArea	0	BW	PW	PH-2*BW
Footing						

Page	NoArea	NoArea	0	PH-BW	PW	BW
LeftMargin						
Body	Middle	NoArea	0	0	BW	PH-2*BW
Middle						
Body	RightMargin	Main	BW	0	PW-2*BW	PH-2*BW
RightMargin						
Body	NoArea	NoArea	PW-BW	0	BW	PH-2*BW
Main						
Middle	FootNote	NoArea	0	0	PW-2*BW	PH-2*BW
FootNote						
Middle	NoArea	NoArea	0	PH-2*BW	PW-2*BW	0

TitlePage

Page						
NoArea	NoArea	Heading	0	0	PW	PH
Heading						
Page	Body	LeftMargin	0	0	PW	BW
Body						
Page	Footing	NoArea	0	BW	PW	PH-2*BW
Footing						
Page	NoArea	NoArea	0	PH-BW	PW	BW
LeftMargin						
Body	Middle	NoArea	0	0	BW	PH-2*BW
Middle						
Body	RightMargin	NoArea	BW	0	PW-2*BW	PH-2*BW
RightMargin						
Body	NoArea	NoArea	PW-BW	0	BW	PH-2*BW

4.16.3 Actions of the layout procedures

In addition to the parameters described above, the layout procedures also read a small number of parameters which modify their behaviour. Both of the procedures read the value of `FinishonEven`, which takes a boolean value. The parameter determines whether a blank page will be generated after a layout procedure has been called, in order to ensure that each portion of the document starts on an odd page. The parameter is set by the statement

```
@Layout (FinishonEven = True)
```

The default value is `true`. Other parameters are specific to the layout procedures, and are described in the corresponding section.

4.16.3.1 The Default layout routine

This procedure can be passed a number of parameters, which determine the way it operates. The statement

```
@Default (HeadingFirst = False, Expand = False)
```

specifies that the procedure will not place a heading on the first page that it produces, and that it will not stretch the leading between lines to get the lines to fill the `Main Area`. These are the default settings of the parameters. (There are other parameters taken by the statement, but these are experimental at present, so you should use named parameters only.) The procedure operates by extracting slugs and boxes in sequence from the principal galley, and forming them into collections. All the slugs of a collection must be placed into the same page. Normally only one slug occurs in a collection; however, several slugs will occur if it is necessary to avoid a widow or orphan, or if an annotation or footnote is associated with a slug, and the galley in which the annotation or slug occurs is in the associated galleys. In this latter case the slugs of the annotation or footnote are placed at the bottom of the page, and the boundaries of the `Main` area and `FootNote` area are adjusted. The procedure makes two attempts to fit a box with a negative `Need` parameter into a page: first it tries to fit the box into the current page; and if this fails, it tries to fit the box into a new page. If this also fails, the box is treated as though `Need` had been set to 0.

When a page is full, the next box from the `PageHeading` and `PageFooting` are placed into the `Heading` area and the `Footing` area, except on the first page if `HeadingFirst` is true, when only the page footing box is placed. Thus if you are using alternating titles and footings on even and odd numbered pages, you must remember that the footing starts on the odd numbered page, the heading of the even numbered page. I ought to fix that.

4.16.3.2 The `TitlePage0` layout routine

This procedure produces a page laid out according to information that is passed to the procedure in the extra environment parameters of the `TitlePage` environment; see below for details. Each of the boxes that occurs in the `TitlePage` is placed at a specified position on the page, and no attempt is made to shift boxes around to make them all fit on the page without overlapping. If there are cross references into other galleys, these are ignored (almost a matter of principle — I dislike footnotes in titles). If an abstract occurs, a heading slug containing the centred word `Abstract` is created.

The position of each of the boxes is specified in the environment parameters of the `TitlePage` environment. These values, measured as vertical distances from the top of the page, are placed in the `Tabulations` entries of the environment, and picked out by the `TitlePage0` procedure. The values can be changed by passing parameters to the environment. See section 4.5.2.2 for details.

(There is also a `titlepage` statement, but the parameters it takes invoke features which are still experimental.)

4.16.3.3 The `Contents0` layout routine

This routine is used to lay out the table of contents; it is parasitic on the layout routine in the `DefaultPS` position of the layout vector, which it calls when it has loaded up the `Contents` galley with information extracted from the representation. Thus any parameters that are applied to the `default` routines will also apply the pages created for the table of contents. More details are given in section ****.

4.17 Macrogenerator

Mint has a macrogenerator front-end which feeds the lexical scanner. The macrogenerator is intended to be used for simple textual replacements and not as a general computational facility — other facilities in Mint provide the features obtained using the macrogenerator in Scribe.

The macrogenerator is modeled on that described by Strachey [Computer Journal, 1963], though many cosmetic changes have been made. These are described below.

In addition to the straightforward textual replacements, the macrogenerator plays three other essential roles: it is used to access system information, such as the time of day, or the current source input file; it is used to pre-process several statements that change the actions of Mint, for example `AssocFont`, `NewGallery` etc; and finally it is used as an integral part of the *Note* facility, within which the bibliography facility acts as a subset.

One word of warning. The balance between the macroprocessor and the error-correcting parser has been a little delicate in the past, and it may well still be so.

4.17.1 Input conventions

A macrogenerator command has the following appearance:

```
@IsEq (abcd, efgh)
```

with the command identifier preceded by @, and the parameters enclosed in brackets and separated by commas. In agreement with the conventions, any pair of bracketing characters may be used: (and), [and], { and }, < and >, " and ", ' and ', and ` and ` . Spaces may follow the macro identifier (but not the @), and may precede and follow the arguments. The sequence of characters @~ will cause the macrogenerator to ignore all characters up to the end of the line, or to the end of the input.

A macro that takes no arguments may be written as, for example

```
@newline()
```

Alternatively, if the next character after intervening spaces is a comma, or an equal, or the current closing bracket of a surrounding macro call, or is the end of file, newline or newpage, then the call can be abbreviated to

```
@newline
```

This convention catches most of the lax macro call conventions of Scribe, but some do get past.

Note that the parameters do not need to be quoted. If not quoted, the macrogenerator will interpret commas, equals, @ symbols, and the close bracket of the surrounding macro call, so the argument should not normally contain any of these; to quote a string the quote used is @" (argh!) with the string to be quoted enclosed in any of the normal brackets. The only interpretation that is performed on a quoted string is to look for the closing bracket; thus some care is needed when using nested quotations. The macrogenerator strips off the quote; subsequent rescanning during macro expansion will cause the control characters (@, comma, equals, etc.) to be acted upon. You need to be aware of how frequently a string will be scanned internally, therefore, before the macrogenerator emits its characters to the lexical scanner — a good reason not to use the macrogenerator as a general computational tool.

In addition to providing for positional parameters, the macrogenerator also provides named parameters, for example

```
@AssocFont (Galley = Main, FontSize = N, FaceCode = F0, FontName = Nonie10i)
```

Named and positional actual parameters can be mixed; parameters are bound from left to right, with the parameter number of positional parameters being determined by which parameter it is in the argument list. It is possible to assign several values to the same formal parameter; the last one bound is the value passed to the macro expander. Leading and trailing spaces are stripped from both the formal and actual arguments (even if quoted, alas; I should fix that), and the case of the characters of the formal parameters is not significant.

Default values for arguments that are omitted from the parameter list may be specified at the time the macro is defined. See section 4.17.2 for more details.

In order to model quite closely the Scribe conventions about text macros, which do not interpret any characters except for the closing bracket, the Mint macrogenerator adopts the following rigid conventions. If the identifier before the equals symbol is not a formal parameter of the macro, then it is incorporated as a part of the actual parameter, together with the equals symbol and any surrounding lexographic display symbols. In addition, if a positional convention is used for expressing the actual parameters, then all the characters that remain in the actual parameter list after prior parameters have been satisfied are incorporated into the last actual parameter. Thus, given that `Comment` is a one parameter macro, the macro call

```
@comment(This is a string that contains an = symbol, in addition to a comma)
```

will place the whole of the argument string into the only parameter.

4.17.2 Defining macros

Macros are defined in one of two ways — using `Form`, which creates the macro template, and using `Defer`, which allows a variety of forms to be associated with a macro. Discussion of `Defer` will be deferred to section 4.17.4. (There is a third way, using `EDef`, but that can only be used in the `maths` environment, so it will be discussed there.)

`Form` is a macro that takes three arguments: the identifier of the macro to be defined; the parameter list, with default values if needed; and the body.

```
@Form(id = MyMacro, params = X, body = Just output this string)
```

This defines a macro `MyMacro`, that takes one parameter, `X`, whose body comprises the string `Just output this string`. This macro does not use the value of its parameter, so it will `Just output this string` for all the following calls

```
@MyMacro() @MyMacro(X = Foo) @MyMacro(Bah)
```

To access the value of a parameter, the macro call

```
@Value(Id = X) or @Value(X)
```

is used. `Value` performs an inside-out search down the static chain (in good old algebraic language tradition) to find a macro definition with a formal macro parameter with identifier `X`. Thus you could write

```
@Form(MyMacro, X, @"{Just output @value(X)}")
```

(Note the careful choice of brackets).

The `Params` parameter of the `Form` macro takes a string which is then picked apart to find the identifiers of the formal parameters, and the default values. Normally this string contains commas and equals, so it is necessary to quote it. Its general form is

```
@Form(MyMacro, Params = @"(P1, P2 = default1, P3 = default2), Somebody)
```

The `Params` argument is analysed in the same way as a macro call. Thus this call specifies a macro with three parameters, whose formal identifiers are `P1`, `P2` and `P3`, with the second and third parameters having defaults `default1` and `default2`. The defaults will be used if `MyMacro` is called without a `P2` or a `P3` parameter.

4.17.3 Access to system values

The `Value` call provides access to macro parameters down the static chain. At the end of the static chain are the parameters of a macro call within which Mint can be considered to have been invoked. The actual parameters of this macro call are various useful system values, for example the time of day and version number. Section 4.18.2 lists the values accessible in this way.

4.17.4 Deferred Macros

Macros may be defined so that calls of the macro occur immediately, or they may be defined in such a way that calls are *deferred*. When a macro call is deferred, it is not evaluated immediately, but instead it is

saved, and can later be *reinvoked*. This subsequent reinvocation may use any (non-deferred) macro definition to interpret the call.

To specify that macro calls on macro `Book` are to be delayed, the call

```
@defer(Book)
```

is made. Any subsequent call on the `Book` macro will result in the call being parcelled up and placed on a list (which, for reasons which will become apparent later, is called the *Plagiarist List*). Calls of the `Book` macro will normally have several parameters; one of them must be a `CodeWord` parameter, written either as an explicit named parameter, or as the first actual parameter.

```
@Book(CodeWord = Knuth68a,  
      Key = Knuth,  
      Author = D.E. Knuth,  
      Title = Fundamental Algorithms,  
      Year = 1968)
```

As many other parameters as one wishes may follow the codeword; they are not interpreted at this stage.

Mint will place this call, without evaluating it, on the `Plagiarist List`. The `Plagiarist List` is a heavy duty data structure, intended to store many hundreds or thousands of delayed macro calls. In particular, it is used to store the bibliographic entries used by the bibliography feature (which can be seen now to use just a general-purpose feature); however, it is also of value in saving the random card-index of notions, notes, quotations, plagiarized cuttings from papers, etc, that make up a part of any academic's intellectual property.

To invoke a delayed macro, the call `ReInvoke` is used, with a codeword as parameter. Assume we have a macro `OutBook`, that takes parameters `Key`, `Author`, `Title`, and `Year`, then the call

```
@OutBook (@ReInvoke(Knuth68a))
```

will be equivalent to

```
@OutBook(Key = Knuth, Author = D.E. Knuth, Title = Fundamental Algorithms,  
         Year = 1968)
```

In this way the notes, comments, etc., can be stored in a free format, to be retrieved when needed using a specific format suitable for the current document. The bibliography feature uses this facility, by generating the reinvocations for the delayed macro calls in the bibliographic database. The macros used for these expansions are specific to the reference style.

4.18 Standard Macrogenerator Facilities

This section lists the predefined macros. We have called predefined macros *statements* elsewhere, and you should look in the relevant sections to see what they do. The only predefined macros described here are those termed *special macros*, which are concerned with macro expansion.

4.18.1 Predefined Macros

Predefined macros are conveniently divided into several classes — special macros, which are used to control macro expansion; bibliographic macros, which interact with the bibliography feature; counter macros, which are used to access counters; galley macros, which are used to alter the properties of galleys; and page macros, which interact with page layout and presentations.

Following each macro identifier is the list of parameters it takes. The default value of these parameters is the empty string.

4.18.1.1 Special macros

form	id, params, body
value	id
defer	macro
reinvoke	dm
char	ch
cond	if, then, else
iseq	x, y
include	file
isdefined	param
message	msg
andm	<up to 16 parameters>
orm	<up to 16 parameters>
notm	x
w	word
device	dev

Form, Value, Defer, and ReInvoke have been dealt with in section 4.17. Char takes an integer in decimal or octal notation (octal being indicated by a leading # symbol), or a character preceded by a quote (for example @char(' ,)) or the string sp for space or the string 1n for newline. It creates a character of that value. This character is specially quoted so that it cannot be interpreted by the macrogenerator; thus any character can be created using this macro. (Actually, this is not quite so, but you should regard this as a special feature.)

Cond is a macro that returns its then parameter or its else parameter, according to whether its if parameter is a non-empty string or an empty string, respectively. Note, however, that both parameters are evaluated (sigh!). IsEq returns a non-empty string if its two arguments are the same, after case-folding; IsDefined returns a non-empty string if its argument is non-empty. Andm, Orm and Notm perform the logical operations on their arguments.

Include causes input into the macrogenerator to come from the specified file. The standard search list is used to find the file; if the search fails, Mint will try again with `.Mss` appended, and finally with `.Mint` appended. Includes can be nested arbitrarily deeply. Note that an **Include** does not cause the macrogenerator to start taking input from the file immediately; if it happens to be scanning some macro body when it evaluates the **Include**, it will continue to evaluate the body, until it again needs input from the file, when it will then take input from the new file.

W causes any spaces within its parameter to be treated as words, thus preventing line breaks, and inhibiting the expansion or contraction of the spaces. It turns out that **W** can have anomolous effects on characters created by **Char**, and on certain strings created by **NConv**. That's a bug.

Device does nothing except to issue a warning that the statement does nothing.

Message outputs a message onto the screen, in the lower window.

4.18.1.2 Bibliographic macros

(See section 4.19.) **CiteInCollection**, **Cite** and **BibInclude** take an arbitrary number of parameters.

<code>newcitecollection</code>	<code>collection</code>
<code>citeincollection</code>	<code>collection, <up to 16 citations></code>
<code>cite</code>	<code><up to 16 citations></code>
<code>bibinclude</code>	<code><up to 16 citations></code>

4.18.1.3 Counter macros

(See section 4.22.)

<code>newcounter</code>	<code>id, within, start = 1, change = +1</code>
<code>next</code>	<code>id</code>
<code>set</code>	<code>id, value</code>
<code>alter</code>	<code>id, by</code>
<code>bind</code>	<code>id, value, binding</code>
<code>bindcurval</code>	<code>id, binding</code>
<code>setnext</code>	<code>id</code>
<code>nconv</code>	<code>conv, counter, label</code>
<code>assocconv</code>	<code>conv, nstyle</code>
<code>label</code>	<code>id</code>
<code>tag</code>	<code>id</code>

4.18.1.4 Galley macros

(See sections 4.3 and 4.9.)

<code>assocfont</code>	<code>galley, fontsize, facecode, fontname</code>
<code>substitutearch</code>	<code>device, dfont, dchar, sfont, schar</code>
<code>substitutegap</code>	<code>device, dfont, dchar, size</code>
<code>substituterange</code>	<code>device, dfont, from, to, sfont, schar</code>
<code>substituteicon</code>	<code>device, dfont, dchar, icon, s, x, y, w</code>

makeicon	slugtype, id, file
copyfont	device, dfont, sfont
emptyfont	device, dfont, height, baseline
assocproc	galley, environment, boxroutine, parameters
putchar	dev, c, font, x, y
putline	dev, x, y, w
moveto	dev, x, y
charinfo	dev, font, c, info
mexcharinfo	dev, font, c, info
setmexchar	device, dfont, dchar, y0, yy, yrel
createlfont	device, dfont, sfont
assocfont	galley, fontname

4.18.1.5 Presentation macros

(See sections 4.15 and 4.16.)

crossproofingdefault	targetdevice, viewingdevice, font
crossproofingfont	targetdevice, targetfont, viewingdevice, viewingfont
newline	lines
hsp	length
vsp	length
zsp	page
layout	finishoneven
default	pasteup, headingfirst, expand
titlepage	pasteup
pageparams	pageht, pagewd, borderwd
pagearea	layout, area, backgroundcolour, border, borderstyle

4.18.1.6 Syntax macros

(See section 4.2.)

addrule	nterm, rhs
remrule	nterm, rhs
adddefault	nterm, def
remdefault	nterm, def

4.18.1.7 Computation macros

(See section 4.11.)

setcomplm	cno, sno, val
setcomprm	cno, sno, val
setcompgap	cno, sno, val
setcompfont	dev, cno, sno, val

4.18.1.8 Index macros

(See section 4.20.)

index	key, seckey, rest
indexincollection	collection, key, seckey, rest
newindexcollection	collection

```
indexinclude    <up to 16 collections>
indexparams     params
```

4.18.1.9 Maths macros

(See section 4.27.)

```
mdef            slex, dlex, stype, fcode
edef           id, body
mathsparams    f1, f2, f3, p1, p2, p3, b1, b2, b3, b4, e1
```

4.18.1.10 Border and colour macros

(See sections 4.26 and 4.10.)

```
newpattern     id, w0, s0, c0, w1, s1, c1, w2, s2, c2, w3, s3, c3,
                w4, s4, c4
newborderstyle id, mt1, mtr, mbr, mbl, patt, patr, patb, patl
newcolour      colour, hue, saturation, brightness
```

4.18.1.11 Extra macros

These macros are defined in terms of other macros; they are here for convenience.

```
@form (comment, body,)
@form (ref, lab, @"[@onconv(placestyle,place,@value(lab))])
@form (pageno, lab, @"[@onconv(pagestyle,pageno,@value(lab))])
@form (eqn, lab, @"[@onconv(equationstyle,equation,@value(lab))])
@form (newpage, n,
        @"[@pagecommand(@zsp(@cond(@isdefined(n),+@value(n),+1))])])
@form (libraryfile, file, @"[@include(@value(file).lib)])
```

Note that the `comment` macro is a crock: it does not stop the scanning of its argument, or its interpretation. You can be very surprised by what happens when you use it. I'll fix it soon.

Because of an undesirable feature of the `zsp` statement, the `newpage` macro inserts two new lines before and after its body.

The `LibraryFile` macro is included to help make Mint more compatible with Scribe; there are not yet any library files.

4.18.2 System attributes accessed via @Value

The following values are available by using `Value`.

Date Yields the current date, as provided by the `Perq`. There are not yet any transformations on the format of the date.

Time	Yields the current time of day, as yielded by the Perq. The same comments as those for Date apply.
TimeStamp	The time stamp, comprising the date and time.
Version	The current version number of Mint.
SourceFile	The current source file from which input is being taken. The string returned by this value is the string passed to Include , or specified on invoking Mint, not the full path name.
Manuscript	The root file for the manuscript. The same comments as for SourceFile apply.
Device	The device identifier.
DocType	The document type. This appears as a string with the basic document type followed by a digit, which is the form of the document (zero if none was specified).
BibStyle	The current bibliography style.
IndexStyle	The current index style.

4.19 Bibliographies

Mint supplies a general bibliography feature, similar to that of Scribe, but obtained using delayed and reinvoked macros. Since these are general facilities, Mint has much more ability to handle unusual reference formats, to add additional reference types, and to place references in line.

The principle facilities of the Bibliography feature are provided through other mechanisms. The use of delayed macros has already been mentioned; in addition the citations are introduced into the text using the general cross reference facility (so that if a presentation omits some references, the reference numbers of the references that are present will be changed without having to re-Mint the document).

The bibliography feature is still being polished, so it is not appropriate here to describe all its features. However, the general facilities will be described.

4.19.1 Citation collections

When a citation is made, the keyword of the delayed macro is noted in some *collection*; when the bibliography comes to be placed into the document, it is necessary to specify which collections will contribute to the bibliographic listing. Thus it is possible to send citations to several collections, and place the collections where appropriate — at the end of each chapter, with a general collection at the end of the document, for example. A new citation collection is created by the `NewCiteCollection` statement:

```
@NewCiteCollection (EndofBook)
```

creates a collection.

There is a collection created by Mint: its identifier is `Standard`. Citations will be placed in this collection if it is not specified that they should go into other collections.

4.19.2 Citations

To place citations in the `Standard` collection, use the `Cite` statement:

```
@cite(knuth68a, knuth68b)
```

To place citations in some specific collection, use

```
@CiteinCollection(EndofBook, knuth68a, knuth68b)
```

There is no mechanism yet for including citations in a collection without a reference to them appearing at the point of citation; I'll fix that shortly.

4.19.3 Causing the bibliography to appear

The citations that have been placed in a number of collections can be introduced into the document using the `BibInclude` statement. This takes up to 16 collection identifiers, and constructs the properly sorted bibliography from them.

```
@BibInclude (EndofChapter, EndofBook)
```

The style of the citations, and the appearance of the bibliography, is determined by the citation style. The value of the style is set using the `CiteStyle` parameter in the `make` statement for the document; for example

```
@Make (Report, CiteStyle IEEE)
```

It takes values from `StdNumeric`, `StdAlphabetic`, `CACM` and `IEEE`. The default value is `StdNumeric`.

At the moment the bibliographic macros are still being written. A reasonably complete set exists for `StdNumeric`, but not for any of the others. Given the pressure of other tasks, I'll probably produce them only on demand, and encourage the requester to implement them himself.

4.20 Indexes

The indexing facility in Mint is similar in design to the bibliography feature. In particular, it allows arbitrary parameters to be associated with an index entry, and it allows an arbitrary macro to be used for laying out the entry. In this way it is possible to include notes in the index entry, to provide cross references to other index entries, and to display the location of the entry in any of the conversions available in Mint, as well as to build multi-level indexes. Because the appearance of the index does not need to be determined until it is laid out, the facility provides a useful degree of flexibility for creating the indexes for complex documents. I expect to provide a number of standard macros for laying out index entries in the future.

Now it turns out that the platitudes penned in the paragraph above would be inoffensive if it weren't for the fact that writing useful index macros were very difficult. The difficulties arise because you need to maintain global state between macro calls, and to use a general algorithmic control flow facility, neither of which can be done easily in a macro language. Since there probably will only be a small number of different styles of index needed in Mint documents, it seems reasonable to build a few of them directly into Mint, and provide an extension mechanism to cover unusual cases. The language I have chosen to allow this is Pascal — there is a module in Mint that can be hacked if the facilities I have provided don't suit you. However, I think that the indexing routine I have supplied will satisfy most requirements, and the masochist can still use the macro-driven facility.

Below I will describe the general facilities, and then the indexing routine I have provided in more detail.

4.20.1 Index collections

When an index entry is made, it is associated with a *collection*; when the index entries come to be placed in the document, it is necessary to specify which collections of entries are to be included. Thus it is possible to send entries to several collections, and include the entries selectively throughout the text — at the end of a section, at the end of a chapter or at the end of the whole document, for example. Collections can also be used to classify the index entries if several indexes are needed.

A new index collection is created by the `NewIndexCollection` statement:

```
@NewIndexCollection (Acyclic Organic Compounds)
```

A default index collection is created by Mint: its identifier is `Standard`. Index entries are placed in this collection if it is not specified that they should go into other collections.

4.20.2 Index entries

There are two statements for placing index entries into collections — `Index`, which places the entry into the `Standard` collection; and `IndexInCollection`, which places the entry in some specified collection. In both cases two *keys* can be provided: a *primary key* and a *secondary key*, together with any number of

additional parameters. The keys are used for sorting the index entries; the additional parameters are stored, and are made available when the index finally comes to be made.

The simplest form of index entry just has a primary key. Its form is

```
@index(apples)
```

which will place the entry in the standard collection; and

```
@indexincollection(fruit, apples)
```

which will place the entry in the `Fruit` collection. A secondary key can also be included. For example

```
@index(apples, granny smith)
```

As many more parameters as desired can follow the primary and secondary key parameters (or the collection, primary key and secondary key parameters in the case of `IndexInCollection`); these are not interpreted at this stage, but are saved along with the index entry, in a manner similar to a delayed macro call. For example, if you want to have a “see also” and a “notes” parameter with an index entry, you can write

```
@Index(Key = Apples, SecKey = Granny Smith,  
        SeeAlso = Uncle Ben, Notes = Green and firm)
```

(My advise is to include the formal parameter identifiers if you are using this extended form of `Index` statement. They are `collection` for the collection identifier, `key` for the primary key, and `seckey` for the secondary key.)

4.20.3 Causing the index to appear

The index entries that have been placed in a number of collections can be introduced anywhere in the document using the `IndexInclude` statement. This takes up to 16 collection identifiers; for example

```
@IndexInclude (Fruit, Vegetables)
```

The entries in the collections are first sorted using the primary key. If several entries have the same primary key, they are then sorted using the secondary key. Finally, if several entries have the same primary and secondary keys, they are sorted using the order of appearance of the entries in the document.

After sorting has been performed Mint will then take one of two actions depending on the value of the style parameter `indexstyle` which is set in the parameters to the document’s `make` statement.

- If the style parameter `indexstyle` has been set to `macro`, then a call of the macro `OutIndex` is created for each entry, and is injected into the input stream. `OutIndex` is passed all the parameters in the `Index` statement, together with a label parameter (whose formal identifier is `Lab`). The label parameter gives the identifier of the label that was attached to the document at

the point where the `index` statement was made. It is expected that a definition of `OutIndex` will have been provided by the Mint user; the macro has complete freedom to handle the call as it wishes.

For example, one of the macro calls that will be generated by the `IndexInclude` statement above will be

```
@outindex(key=Apples, seckey=Granny Smith, lab=ix0008,  
          seealso=Uncle Ben, notes=Green and firm)
```

(The label identifier is created by Mint; the Minter should avoid creating labels comprising the letters `ix` followed by four digits.)

- If the style parameter `indexstyle` has been set to `style1`, Mint will instead call the Pascal indexing routine that I have provided. This routine accumulates a number of items of information that are passed to it, transforms the information, and finally emits it in a style appropriate for making two-level indexes. I describe the actions of the routine in more detail in the next section.

(For those who are interested, I expect eventually to have `style2`, `style3`, etc., each providing different forms of index.)

The default index style is `style1`.

4.20.3.1 The `Style1` indexing routine

This routine expects the index entries to have a `key`, `seckey` and `style` parameter; other parameters are ignored. The output comprises an entry for each primary key, in alphabetical order, with the entries for each secondary key associated with a particular primary key set out in alphabetic order, and with the document locations following the secondary key in reference order. The `verse` environment is used to set out each primary index entry, so that the second and subsequent lines that an entry occupies are indented more than the first; a new `verse` environment is started for each change of initial letter. In this way it is possible to control the amount of space between entries, and between collections of entries that differ by their initial letter. The index for this document was produced using the `style1` index routine.

The `style` parameter determines how the reference to the entry will appear in the index. Mint generates a call to a macro taking one parameter to perform the conversion. For example, if the `style` parameter is `pageno`, then the entry will appear as a page number; if it is `ref` it will appear as a section number. You can provide your own conversion macros if you wish. For example, I obtained the bold page number entries by defining

```
@form (BoldPageNo, X, @""@b{@pageno(@value(X))})"
```

and then specifying my index entry like

```
@Index (Box Parameters, Definitions, Style = BoldPageNo)
```

If there is no `style` parameter, Mint uses `pageno` to do the conversion.

In general you will want to set the box environment parameters of the verse environment, to change its size or change the fonts used. The statement

```
@IndexParameters (@""Width 3in, TabClear, TabSet 1.75in,  
ExtraLeftMargin 2.25in")
```

will set the appropriate parameters (these were the ones used for the index in this manual; there was also a `modify` operating).

4.21 Prefixes and postfixes

The first slug of a box can have a prefix string generated for it. This prefix string appears in the slug before any of the input from the manuscript. Normally it consists of numbering information that gets generated by Mint, though there is no need for this to be so. In the current version of Mint, prefixes are bound fairly tightly into the system, and are specified with the syntax, though there is no reason for this, and a more appropriate place would be to specify the prefixes along with the procedures and the environment parameters in the procedure family of a galley. This change will be made shortly. The `maths` environment has a postfix string generated for it, but I'll call all inserted string *prefixes* to avoid confusion.

Prefixes are not source strings introduced early in the processing of the input for a (box) procedure; the reason why this is not so is because Mint keeps a tight control over the processing of the input — the need for a prefix can only be recognized after syntactic analysis has been performed, and introducing an arbitrary string at this point could destroy the formal properties of the output from the parser. While this implies a lack of generality as compared to Scribe, the counter facility in Mint appears to allow all the meaningful uses of prefixes (or at least all those I can think of). The loss is a certain flexibility, since the prefixes essentially have to be programmed into the system using a language different from that used for the manuscript. I've worried about this for some time, and see a way of hacking a route out, but I need to put more thought into it.

Since prefixes are not programmable by the casual user, this section simply presents those that are available. Eventually it will be possible to read prefixes in from some data base, thereby overcoming much of the inflexibility.

4.21.1 Standard prefixes

Several prefixes are defined in Mint. A general description of them follows.

Place Prefix

Place prefixes are used for the standard section environments. They comprise a definition of a new cross reference point (see section 4.22), followed by an applied occurrence of the appropriate counter (Section, SubSection, Paragraph), using the `PlaceStyle` conversion style. When the cross

	reference is resolved (at page layout time), the applied occurrence will appear in the current <code>PlaceStyle</code> , in the font size and face code of the section slug.
Figure Prefix	A figure prefix is used to create the numbers for the captions of figures. It comprises a definition of a new cross reference point (see section 4.22), the word <code>Figure</code> , followed by an applied occurrence of the <code>Figure</code> counter, using the <code>FigureStyle</code> conversion style. The prefix string appears in the prevailing <code>b</code> face code.
Table Prefix	A table prefix is used to create the numbers for the captions of tables. It comprises a definition of a new cross reference point (see section 4.22), the word <code>Table</code> , followed by an applied occurrence of the <code>Table</code> counter, using the <code>TableStyle</code> conversion style. The prefix string appears in the prevailing <code>b</code> face code.
Chapter Prefix	The chapter prefix is used to create the chapter prefix for those document styles that have chapters in their section environments. It comprises a definition of a new cross reference point (see section 4.22), the word <code>Chapter</code> or the word <code>Part</code> , followed by an applied occurrence of the <code>Chapter</code> counter, using the <code>ChapterStyle</code> conversion style. The chapter title follows on the next line.
Appendix Prefix	The appendix prefix is used to create the appendix prefix for those document styles that have appendices in their section environments. It comprises a definition of a new cross reference point (see section 4.22), the word <code>Appendix</code> , followed by an applied occurrence of the <code>Appendix</code> counter, using the <code>AppendixStyle</code> conversion style. The appendix title follows on the next line.
Copyright Prefix	The copyright prefix is used to create the copyright notice on title pages. It comprises the word <code>Copyright</code> , followed by the copyright symbol, followed by the year. Eventually a counter will be set with the year value, and a conversion style associated with the counter, so that the year can be output in any of the conversion styles.
Equation Postfix	The equation postfix is appended to the end of a formula created by the <code>maths</code> environment if it has a label parameter. It comprises an equation number (see section 4.27.3.2) enclosed within parentheses.

The prefixes that are available are as follows. The counter associated with the prefix is given in the second column.

<code>PrefixChapter0</code>	<code>ChapterNo</code>	A chapter prefix using <code>Chapter</code>
<code>PrefixChapter1</code>	<code>ChapterNo</code>	A chapter prefix using <code>Part</code>
<code>PrefixAppendix0</code>	<code>AppendixNo</code>	A appendix prefix
<code>PrefixSection0</code>	<code>SectionNo</code>	A place prefix
<code>PrefixSubSection0</code>	<code>SubSectionNo</code>	A place prefix
<code>PrefixParagraph0</code>	<code>ParagraphNo</code>	A place prefix
<code>PrefixAppendix1</code>	<code>AppendixNo</code>	A place prefix
<code>PrefixAppendixSec0</code>	<code>AppendixSecNo</code>	A place prefix
<code>PrefixFigure0</code>	<code>FigureNo</code>	A figure prefix
<code>PrefixTable0</code>	<code>TableNo</code>	A table prefix
<code>PrefixCopyrtN0</code>	<code><none></code>	A copyright prefix
<code>PostfixEqn0</code>	<code>Equation</code>	An equation postfix

See section 4.4.3 for the standard associations.

4.22 Counters and Labels

In order to cross reference one part of a document from another part, *tags* need to be attached to parts of the document, and a mechanism provided to refer to the tag. (The term *tag* is not being used here in the sense that Scribe uses it. Consider a tag as a defining occurrence of some internal label, that causes Mint to record the location of the tag — the slug or box in which the definition of the tag occurs).

Internal tags are a poor facility to help the reader of a document find his way around it, since he generally has a notion of different classes of information in the document, such as chapters, sections, figures, formulae, etc. It is helpful to have each of these use distinct labelling schemes, so that one can refer to chapter one, figure five, etc., independently of the numbering of the other classes of information.

Several labelling schemes could be conceived; however, to generate label values automatically, it is useful to employ *counters*, each associated with a different class of information. Thus the chapter counter, used to generate chapter numbers, can be independent of the page counter, used to generate page numbers.

Mint provides a general scheme to allow counters to be created, and associated with classes of information. In addition the values of the counters can be displayed in a variety of different styles. This section describes the facilities provided.

4.22.1 Overview of Counters

Assume that several *counters* have been defined (more details are given below on how to define a new counter; several are predefined in Mint). A counter has an integer value, that can be changed (usually simply incremented) during the processing of the manuscript. This may occur automatically (such as is the case with the counter associated with figures), or it may be changed explicitly. Every counter, at some point in the processing of the manuscript, has some value, and the collection of values of the counters is called the *Counter Contour* at that point in the document.

A snapshot of the contour is taken when a *label* is defined, using the `Label` statement. It is possible to extract the value of any of the counters in the contour, and to introduce its value, converted in any of a number of conversion styles, by referring to the label in the appropriate way. This is the case not only for the standard counters used by Mint, but also for any of the counters declared by the user. Furthermore, the applied occurrences of counters are only resolved at page layout time, so that forward references to labels can be handled without having to re-Mint the document⁷.

⁷ Even better, the *style* with which the counter value is converted need not be specified until page layout time, so that different presentations can use different conversions. Mint re-evaluates the contour for each presentation, on the expectation that the presentations may not have the same contents, and hence the same counter-changing statements.

Since several copies of slugs that define labels may get taken during page layout, Mint has mechanisms to ensure that the final appearance of a document is as though several (unique) labels have been defined.

In addition to counters from which it is possible to extract a single value, you need to have *pseudo-counters*, which yield a composite value, for example the value of the chapter counter followed by the value of the equation counter. Several such counters are built into Mint, with the hope that they provide all the standard needs. If they don't satisfy your requirements, you will have to hack new ones using macros.

In review, then, Mint's facilities comprise a means of specifying counters, defining labels, and extracting the values of the counters from the contour associated with a label, in a variety of styles. We consider each of these in more detail below.

4.22.2 Counter manipulations

A new counter is defined by the statement `NewCounter`. This takes a counter identifier, another counter within which the counter will count, and an initial value and increment value. For example

```
@newcounter(TheoremCounter, ChapterNo, 1, +1)
```

defines `TheoremCounter` to start at 1, and be incremented in steps of +1. `TheoremCounter` will be reset back to 1 each time `ChapterNo` is changed. If a counter is required to count independently of other counters, the second parameter should be empty. Counters can start at any positive or negative value, and the increment can be any positive or negative value.

It is possible to set a counter to some arbitrary value at any point in the manuscript by the statement `Set`, and to cause it to be incremented by the increment value using the statement `Next`. For example, after

```
@Set (TheoremCounter, 5)  
@Next (TheoremCounter)
```

`TheoremCounter` will have the value 6. The value of a counter can be altered by the statement `Alter`. For example, after the statements above, the statement

```
@Alter (TheoremCounter, -3)
```

will assign the value 3 to `TheoremCounter`.

Mint rescans a presentation after it has made it, in order to find all the occurrences of `Set`, `Alter` and `Next`, and only then does it remake the slugs that refer to the counters. It is for this reason that a presentation can omit parts of the galleys that have incremented counters using `Next`, but Mint will still cause all the applied occurrences of counters to be consecutively numbered. The only way you can avoid this happening is if you explicitly set a value using `Set` — in this case Mint will cause the counter to have the specified value. You sometimes need to increment a counter and also set it (especially when using the `Binding` conversion); in this case you should use the statement `SetNext`; for example

```
@setnext (authorcounter)
```

If you do not understand this section, then you should always use `next` for counters that are converted using the numeric conversions, and `setnext` for those that use the `binding` conversion, or that need to have the same value in several different presentations.

4.22.3 Labels

A label is defined by the statements `Label` and `Tag`; these take a snapshot of the counter contour, and associates it with the label. The label identifier can be any valid macrogenerator string⁸. For example

```
@Label(Current position)
```

Labels may be defined in any of the galleys. When printing the galleys, the contours associated with them are the contours appropriate for scanning the galleys in some canonical order (the order in which the galleys have been defined); when performing page layout, the contour associated with a label is that appropriate for a sequential scan of the slugs and boxes in the pages. Thus the contours may change, depending on how page layout is performed, and how counters have been incremented in each of the galleys.

4.22.4 Referring to labels

The macrogenerator provides a basic operation for recovering the value of a counter from the contour associated with a label: `NConv`. This statement takes a *conversion*, a counter or pseudo-counter, and a label, and returns the value of the counter in the contour, converted according to the specified conversion. (Conversions are dealt with in more detail below; assume for the moment that we have conversions such as `RomanUC`, to convert to upper case roman, and `Arabic` to convert to arabic numerals).

For example

```
@nconv(Arabic, PageNo, Your Label)
```

will produce the value of the `PageNo` counter in the contour of `Your Label`. Note that this value will depend upon the way that page layout is done, and may differ from one presentation to another.

It is frequently the case that you want to use the current value of a counter, rather than the value at some label. This frequently occurs with the page number that is placed in the heading or footing of a page. It is tedious to have to declare a new label every time, and then refer to it, as follows.

```
@label(Yet_Another_Label)@nconv(Arabic, PageNo, Yet_Another_Label)
```

⁸ Not, it seems, for labels defined in the `maths` box parameters. Sigh!

Sometimes it isn't even possible to do this. To get over the problem, Mint allows the label field in `NConv` to be empty. It then effectively generates a defining occurrence of a unique label automatically, and uses it as an argument to `NConv`. Thus

```
@NConv (Arabic, PageNo, )
```

will produce the current page number converted to arabic numerals.

4.22.5 Conversions

Mint provides a number of in-built *basic conversions*, such as `RomanUC`, a complete list of which is given in section 4.23.1. One of the conversions, `Binding`, allows the user to set up an arbitrary binding between the values a counter can assume and strings.

The identifier that occurs in the `NConv` statement can be either the identifier of a basic conversion, or an identifier that has been associated with a basic conversion by the `AssocConv` statement. For example

```
@AssocConv(TableStyle, RomanLC)
```

specifies that `TableStyle` is currently `RomanLC`. Thus we can also write

```
@NConv (TableStyle, TableNo, )
```

If you do this, you can change the binding between `TableStyle` and the basic conversion at any time (again using `AssocConv`) and thereby change the appearance of the references. It is to allow this flexibility that the standard prefixes are defined in terms of non-basic conversions.

Most of the conversions supplied by Mint are standard conversions from the internal counter value to an external representation of it, in roman numerals, arabic numerals, etc. The `Binding` conversion allows the Minter to establish an arbitrary mapping between internal values and external strings, which can be used to maintain section headings, for example. Every counter has a `binding` conversion that is independent of the bindings of other counters. The statement `Bind` allows an arbitrary string to be bound to some value of a counter. For example

```
@Bind (AuthorCounter, 4, Peter Hibbard)
```

specifies that the `Binding` conversion, applied to counter `AuthorCounter`, will yield the string `Peter Hibbard` if the value of the counter is 4. The value is retrieved using the `NConv` statement. For example, if the current value of `AuthorCounter` is 4, then

```
@NConv (Binding, AuthorCounter, )
```

will yield `Peter Hibbard`.

Usually you do not want to specify some explicit value for the counter — you are content to bind to the current value of the counter. This is done by

```
@BindCurVal (AuthorCounter, Harry Q. Bovik)
```

You should also be aware of the exhortation in the previous section, about the use of `SetNext`.

4.22.6 Undefined labels

Since Mint is intended to be an interactive system, it must be possible to define labels at any time while the document is being created. When a label that has been used is finally defined, Mint will patch up the references to it automatically. Since the philosophy of Mint is that there may always be a label definition coming along later, it does not regard an unresolved label as an error. However, in the non-interactive version this means that warnings are not generated. Since this is somewhat surprising to the casual user, I will fix Mint soon.

When Mint is making slugs that contain counter conversions that it is not yet able to resolve, it places a *filler lexeme* into the slug. The lexeme is normally `?`. When the value becomes known, the slug is remade. In general, Mint's guess for the size of the final counter conversion is close enough that only one slug needs to be remade; however, if several consecutive slugs need to be remade, Mint will do this. Should it be the case that replacing the filler lexeme by the counter conversion causes the number of slugs in the box to change, Mint will throw up its hands and admit defeat. The problem is that if a box changes its size, the page layout may need to be redone, and I'm not prepared to do that in the current non-interactive version. You will know that Mint has problems, since it will tell you, and it will then proceed to overlay the new slug on top of the last one in the box. All is not lost, though; the extra style parameter `Filler` is a (crude) way of changing the size of the filler to be closer to the size of the final lexeme. Try something like

```
@make(Thesis, Filler ***)
```

if you run into the problem, and Mint the document again.

4.23 Standard Conversions and Counters

The following section describes the standard conversions, counters and pseudo-counters that are available in Mint.

4.23.1 Conversions

The following basic conversions are available.

RomanLC	Converts a value into lower case Roman numerals, e.g. <code>vii</code> , <code>xxiv</code> .
RomanUC	Converts a value into upper case Roman numerals, e.g. <code>MCMXLI</code> .
Arabic	Converts a value into arabic numerals. Both negative and positive values can be converted.
AlphaLC	Converts a number into cardinal English numbers, e.g. <code>one hundred and three</code> .
AlphaUC	Converts a number into cardinal English numbers in which the first letter of each word is a capital letter, e.g. <code>Fifty Five</code> .
LetterLC	Converts a number into a lower case letter, <code>1 = a</code> , <code>2 = b</code> , etc.
LetterUC	Converts a number into an upper case letter, <code>1 = A</code> , <code>2 = B</code> , etc.
Binding	Retrieves the arbitrary binding associated with the counter. The binding can be any string.

4.23.2 Pseudo-counters

Pseudo-counters are referred to in the same way as other counters; however, they cannot be `set`, `altered`, or used in any statement except `nconv`. The following are the pseudo-counters.

Place	This pseudo-counter yields the location of the label as a sequence of counter values, for example 4.23.2. Each individual value is converted according to the style in the <code>nconv</code> statement in which the pseudo-counter occurs, except possibly for the first value, which will appear as a letter if the label is in an appendix.
EnvType	This counter yields the environment type of box in which the label occurs, irrespective of the conversion style applied to it. I tried fairly hard to give this pseudo-counter a natural interpretation, given that the Mint error-correcting parser is generating new environments all over. I wasn't too successful; for example, <code>@nconv(arabic, envtype, pscount) @ref(pscount)</code> yields default 4.23.2.
citation	I must confess I've forgotten what this does.
Equation	This yields the equation number, represented as a chapter number, followed by the equation number within the chapter. See section 4.27.3.2 for more details.

4.23.3 Non-basic conversions

The following conversions are defined for the use of the standard counters. Their initial bindings are shown.

AnnoteStyle	Arabic
PartStyle	Arabic
PageStyle	Arabic
PlaceStyle	Arabic
ChapterStyle	AlphaUC

SectionStyle	Arabic
SubSectionStyle	Arabic
ParagraphStyle	Arabic
TableStyle	Arabic
FigureStyle	Arabic
AppendixStyle	LetterUC
AppendixSecStyle	Arabic

4.23.4 Counters

The following counters are defined for use in the standard prefixes (section 4.4.3, 4.21), the standard annotations (section ***), and in the standard presentation (section 4.23.1). The counter identifier, the counter within which it counts (if any), the initial value and the increment are shown.

4.23.4.1 Counters common to all document types

PartNo		1	+1
PageNo	PartNo	1	+1
FigureNo		1	+1
EquationNo		1	+1 ⁹

4.23.4.2 Counters in document types with footnotes and annotations

AnnoteNo		1	+1
----------	--	---	----

4.23.4.3 Counters in document types that have chapters

ChapterNo		1	+1
SectionNo	ChapterNo	1	+1
SubSectionNo	SectionNo	1	+1
ParagraphNo	SubSectionNo	1	+1
AppendixNo		1	+1
AppendixSecNo	AppendixNo	1	+1
EquationNo	ChapterNo	1	+1

4.23.4.4 Counters in document types that have sections

SectionNo		1	+1
SubSectionNo	SectionNo	1	+1
ParagraphNo	SubSectionNo	1	+1
AppendixNo		1	+1
AppendixSecNo	AppendixNo	1	+1
EquationNo	SectionNo	1	+1

⁹ In document types with sections or chapters, the equationno counter counts within the corresponding counter.

4.24 Miscellaneous layout statements

This section describes several slug, page and document layout commands that are not conveniently described elsewhere.

4.24.1 Spacing statements

<code>@hsp</code>	This takes a single parameter, which is a horizontal distance. If the statement occurs in an environment other than <code>@pageoffset</code> the specified amount of space is left in the slug in which it occurs. If the distance is negative, backspacing within the slug occurs, and overprinting may occur.
<code>@vsp</code>	This takes a single parameter, which is a vertical distance. If the statement occurs in an environment other than <code>@pageoffset</code> then a slug of the specified size is inserted into the box; the slug is otherwise empty. The distance may be negative, in which case lines may overwrite each other. This statement will produce unusual results if it occurs on a line that contains other characters.
<code>newline</code>	This takes a single parameter, which is the number of slugs to terminate. If the parameter is absent or is empty, then the current slug is terminated. <code>@*</code> has the same effect as <code>@newline(1)</code> . <code>newline</code> is a macro which calls <code>zsp</code> , which is the basic statement controlling the generation on new pages. <code>newline</code> is defined in section 4.18.1.11.

The effects of `Hsp` and `Vsp`, if they cause the slug or box to exceed its normal size, are unpredictable (or rather, I don't know what will happen).

4.24.2 Page commands

Commands are passed to the page layout procedures via the `OddsandSods` galley, which basically allows several commands to be delayed. These commands allow for running headings and footings to be inserted on pages, for pages to be offset to allow binding margins, and for page skips to be made.

4.24.2.1 Page headings and footings

The `PageHeading` and `PageFooting` environments allow any of the standard terminal environments to be nested inside them. The cross references that these environments leave in the `Main` galley are picked up by the `Default0` layout procedure, and the boxes within them are then placed in the heading and footing area of the page as it is completed. The boxes are selected sequentially, and the `PageHeading` and `PageFooting` box is restarted after it is complete. In this way several headings and footings can alternate at any cycle length.

Note that if a slug contains a definition of a label (either explicitly, or implicitly through using `NConv`

with an empty label parameter), Mint will create new, unique labels to ensure that cross references are correct. Thus, to obtain a page number at the bottom of each page, the statement

```
@pagefooting(@pageno())
```

suffices.

4.24.2.2 Page offsets

The `PageOffset` environment feeds slugs to the layout procedures in a manner similar to that for `PageHeading` and `PageFooting`. Each slug is scanned for `Vsp` and `Hsp` statements. If either is found, the page is offset by that amount. Thus it is possible to shift pages either horizontally or vertically, and to set up cycles. For example, pages that are to be reproduced back-to-back, and stapled in the top left-hand corner, can be adjusted by

```
@begin(PageOffset)
  @Hsp(+0.25in)@Vsp(+0.25in)
  @Hsp(-0.25in)@Vsp(+0.25in)
@end(PageOffset)
```

4.24.2.3 Page skips

The `PageCommand` environment is a general way of passing information to the layout procedures. Currently they only recognize one item in the slugs within the environment, the `Zsp` statement. `Zsp` takes one parameter, which is either an unsigned number, or a signed number. If an unsigned number, the layout procedure will skip to that page number in the current part, if the page has not already been finished; if a signed number, the layout procedure will skip that number of pages. To skip one page, the statement

```
@PageCommand (@Zsp (+1))
```

is used. The `Newpage` macro provides a convenient way of using this statement.

4.24.3 Tabulations

Tabulations are a part of the Mint environment parameters, and are set and cleared when the environment starts; thus the casual use of tabulations that Scribe encourages is not permitted.

To clear the standard tabulations of an environment, the `TabClear` environment parameter is used. Tabulations are set using `Tabset` followed by a horizontal distance. Alternatively tabulations can be set equidistant along the box by `TabDivide`, which clears all previous tabulations. The environment parameters are scanned from left to right, so that to set a pair of tabulations, one does

```
@begin(description, tabclear, tabset 8 ems, tabset 18 ems)
```

Tabulations can be set dynamically by the use of @†; this adds another tabulation to the collection for the environment. To move to a tabulation stop, @\ is used.

At most 10 tabulations can be set for an environment in the box parameters. Any number of additional tabulations can be set using @†.

4.24.4 The Align environment

The Align environment treats @\ differently from the way other environments treat it. (The maths environment treats tabulations in a similar way to the align environment; see 4.27.) On encountering a @\, Mint searches backwards in the slug until it finds a space, and it then stretches the space so that the tabulation occurs at the correct position. Thus if the tabulation occurs in the middle of a sequence of characters, all the characters are shifted to the right. This allows vertical lists to be aligned around arbitrary points. For example,

```
@newpattern(line2,2,1)
@newborderstyle(width2,n,n,n,n,line2,line2,line2,line2)
@begin(table, width 5.5in, borderstyle width2)
@begin(caption, border 0.1in, borderstyle width1)
Personal Expenses Claimed against Income
@end(caption)
@begin(align, tabclear, border 0.1in)
    @u(Nature of exp@^ense claimed)                @u(Amount c@^laimed)

@w(Depreciation @\- Fabric) $125@
@w(Depreciation @\- Furnishings) $95@.50
@w(Heating @\- Gas) $26@.25
@w(Heating @\- Electricity) $57@.0
@end(align)
@end(table)
```

produces the following effect

Table 11. Personal Expenses Claimed against Income	
<u>Nature of expense claimed</u>	<u>Amount claimed</u>
Depreciation - Fabric	\$125
Depreciation - Furnishings	\$95.50
Heating - Gas	\$26.25
Heating - Electricity	\$57.0

The Align environment also allows you to centre text around some tabulation. The marker @< causes Mint to search backwards in the slug to find the preceding tabulation, and it then adjusts the spacing so that the text between the two is centred about the tabulation. To help in the visual appearance of the .Mss file, I have provided the tabulation mark @>, which has the same effect as @\ . Thus you would normally write

```
@>Text to be centred@<
```

In order to simplify the use of @< and @\ in the align environment, Mint inserts a space of width zero after them; this allows you to write @>x@<y@\.

A comprehensive example of the use of the Align environment is provided by the PageHeading and PageFoot ing environments that have been used for the heading and footing of this manual. They are

```
@form (chap,x,@"@"Chapter(@value(x))
  @begin(pageheading)
    @begin(align,border=0.1in,borderstyle=width1,tabdivide 2)
      @pageno() @>@b(@value(x))@< @ovp( @+(@w(Peter Hibbard))@\\)@-( @w(Mint
User Manual))@\\
    @end(align)
    @begin(align,border=0.1in,borderstyle=width1,tabdivide 2)
      @ovp(@+(@w(Peter Hibbard))@-(@w(Mint User Manual)) @>@b(@value(x))@<
@pageno()@\\
    @end(align)
  @end(pageheading)
  @begin(pagefooting)
    @begin(align,tabdivide 1)
      @s(@value(sourcefile))
    @end(align)
    @begin(align,tabdivide 1)
      @s(@value(sourcefile))@\\
    @end(align)
  @end(pagefooting)
")
```

4.24.5 The Describe¹⁰ environment

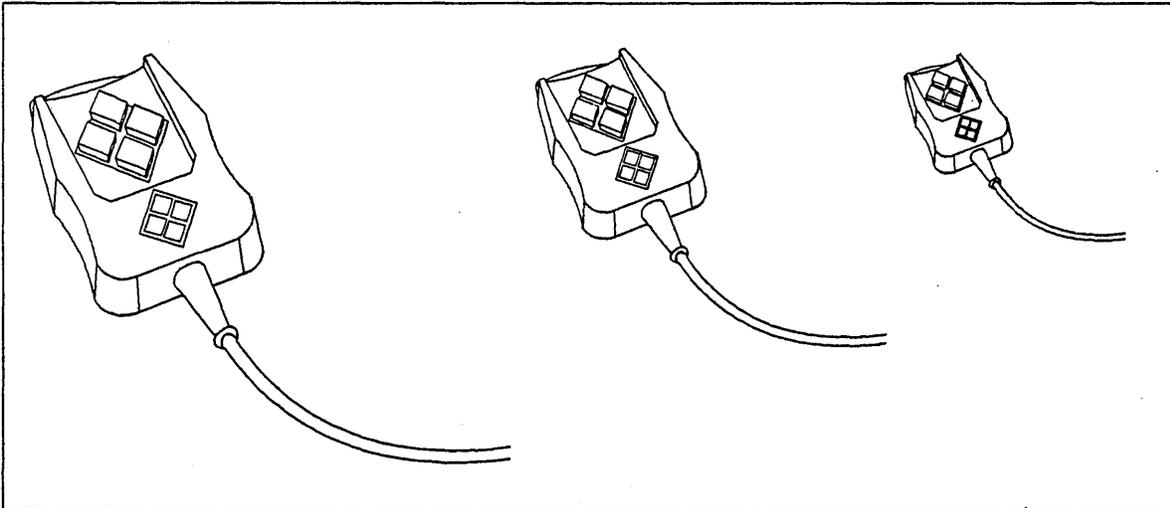
Sometimes it is necessary to place two or more environments side by side. The commentary facility provides one such example. Mint provides a general facility for laying out environments adjacent to each other, so that you can select the widths of each of the environments. The commentary feature is, in fact, a degenerate case of the describe environment, and the default values of the several parameters that describe takes allow it to be used for the description environment as well.

The number of environments that are placed adjacent to each other, and their widths, are determined by the tabulations of the describe environment. For example, if you wanted to place three DP drawings next to each other, you could do

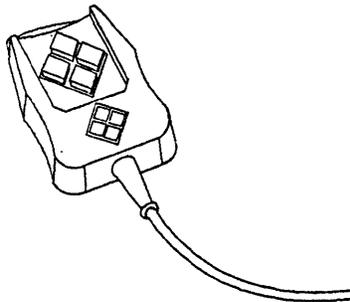
```
@begin(describe, tabclear, tabset 2.75in, tabset 4.75in, border 0.15in,
  borderstyle width1)
  @dp(@include(mouse.dp))
  @dp(@include(mouse.dp))
  @dp(@include(mouse.dp))
@end(describe)
```

¹⁰ Pronounce it as des'cribe, or de'scribe, as you wish.

which produces the following



But there is no need
for the environments
to be the same.
For example,
you can flush one
environment
to the right
and another



to the left.
In fact, I could
have used an
itemize,
enumerate,
or even another
describe
here, to produce a
number of effects.

One particular use of the `describe` is to lay out tables; those in section 4.4.2 have been produced in this way.

If you do not specify the tabulations explicitly, the `describe` environment provides one tabulation only, at a quarter of the distance across the box. This is the same distance as the items occur in a `description` environment.

4.25 Cross proofing

Mint provides a general mechanism for viewing the output intended for one device (the *target* device) on another device (the *viewing* device). It does this by scaling the page size for the target device so that it fits on the viewing device's page. It then computes where to place each character and line intended for the target device on the scaled image of the page on the viewing device.

For lines generated as box outlines, or by the `Plot` and `DP` environments, the scaling can be done with reasonable accuracy; however, Mint is not able to scale fonts, and for this reason must select a font on the viewing device to replace that on the target device. In general, the result will not be as pleasing or readable as it would be if the document were printed on the target device, but since each character is in the appropriate relative position, cross proofing can be of great value in reducing the turn-round time for documents, especially where the layout must be carefully controlled.

Mint selects a font to replace the target device font based on choices made by me; however, these choices can be changed by the user. Currently I use `Gacha9` on the `Perq` to view all fonts for the `Dover`, and `Gacha12` on the `Dover` to view all the fonts for the `Perq`.

The following statements may be used to alter this meagre state of affairs. The statement `CrossProofingDefault` will alter the default font:

```
@CrossProofingDefault (Perq, Dover, TimesRoman10)
```

specifies that when the `Perq`, as target device, is viewed on the `Dover`, as viewing device, then `TimesRoman10` will be used as the default font. A specific replacement can be made also:

```
@CrossProofingFont (Perq, Gacha9, Dover, Gacha8)
```

specifies that if some character for the `Perq`, as target device, is in `Gacha9` font on the `Perq`, then it is to appear as `Gacha8` on the `Dover`. (This cross proofing replacement occurs after all character substitutions). One can build up a reasonable collection of cross proofing font pairs.

4.26 Borders and Border Styles

Mint provides a facility for drawing a border round any box and any page area. This facility may be used to set off pieces of text, to frame diagrams, and to contain tables.

There are two abstractions provided — the *border* and the *border style*. Every box has a border between the outside of the box and the inside; the margins of the slugs coincide with the inside of the box; see figure ***. The size of the border is specified in the environment parameters, and normally is zero. To set it non-zero the environment parameter `border` is used:

```
@begin(figure, border = 0.5cms)
```

The border style of a box specifies the appearance of the border; it is possible to construct different border styles. The border is drawn on the inside of the box; if the width of the border is not enough to contain the border style, then part of the box contents may be overwritten. The border style is specified as an environment parameter. Normally it is equal to `NoBorder`, but it may be set to other values:

```
@define(boxed = description, border = 0.05inches, borderstyle = width1)
```

Border styles are built up using lower level abstractions. These are described below.

4.26.1 Border Styles

To create a new border style you first of all create some *patterns* by collecting together several *lines*; then you collect the patterns together to create the border style. Below I describe how you go about this task.

4.26.1.1 Lines

Line styles are built into Mint (at present). There are four styles provided; you refer to these patterns by their number:

Line style 0	This is an empty line.
Line style 1	This is a solid black line.
Line style 2	This is a dashed line in which long black and short blank spaces alternate.
Line style 3	This is a dashed line, in which a long and a short black line are alternated, and separated by a short blank space.

4.26.1.2 Patterns

From the primitive line styles you build up a pattern, using the statement `NewPattern`. This takes up to five triplets of parameters, which specify the width of the line style, the line style number, and the colour of the line to be used to create the pattern. Thus a pattern comprises up to five different adjacent line patterns. The `NewPattern` statement associates an identifier with the pattern.

```
@newpattern(baroque,8,1,black,2,0,transparent,1,1,black,2,3,black,1,1,black)
```

This specifies the pattern `Baroque`, comprising of $8/100^{\text{th}}$ inch of solid black, $2/100^{\text{th}}$ inch that are blank, and a $2/100^{\text{th}}$ inch of dashed line encased by two $1/100^{\text{th}}$ inch black lines. The pattern specifies the appearance of a pattern, from the outside towards the inside of the box. (See section 4.10 for a description of colours.)

4.26.1.3 Border Styles

A border style is built up from four patterns, one for each edge of the box, starting at the top and working round in a clockwise direction; and four mitring modes (which take parameters `Y` and `N`). These modes specify how the corners will be mitred, starting at the top left, and working round in a clockwise direction. The border style is specified and associated with an identifier with the statement `NewBorderStyle`. The following example shows how a border style has been created, and the effect of drawing it around the box.

```

@newpattern(Baroque,
            8,1,black,2,0,transparent,1,1,black,2,3,black,1,1,black)
@newpattern(line3,3,1,black)
@newborderstyle(myborder,n,n,y,n,line3,Baroque,Baroque,line1)
@begin(example, width -0.4in, border 0.2in, borderstyle myborder)
@@newpattern(Baroque,
            8,1,black,2,0,transparent,1,1,black,2,3,black,1,1,black)
@@newpattern(line3,3,1,black)
@@newborderstyle(myborder,n,n,y,n,line3,Baroque,Baroque,line1)
@end(example)

Etc -- it's recursive.

```

4.27 Mathematical Typesetting

Mint provides a range of mathematical typesetting facilities which are similar to those provided by \TeX — namely semi-automatic choice of the fonts and of the positions of the characters for a wide variety of common mathematical formulae, and the automatic generation of special characters. Also, in a manner similar to \TeX , Mint allows the rules to be overridden when necessary.

The principal aim in implementing these features in Mint has not been to explore issues about which language facilities are appropriate for mathematical typesetting; but instead it has been to provide a means of obtaining good quality mathematical formulae in Mint documents. For this reason the input language has been chosen for ease of implementation, rather than elegance; however, the semantics that are applied to the input, whereby the fonts and positions of characters are chosen, are intended to be the same as those for \TeX , so that (potentially, at least) any formula that can be typeset using \TeX can be done with comparable effort using Mint, and it will produce identical, or better, output.

A word of caution. Mint's mathematical typesetting facilities are still being developed, and are far from complete or robust. What is available now should satisfy many requirements, but I would appreciate hearing from users who have particular requirements that the current facilities do not satisfy.

In the description below I will be assuming that you have a superficial understanding of \TeX . In the first section I will review briefly the main ideas of mathematical typesetting, and then describe the features in Mint in increasing detail.

4.27.1 Basic Concepts

This review is not intended to be a mathematical typesetting handbook, and many of the concepts are treated superficially. The material here is essential for an understanding of Mint's facilities, though.

Mathematical formulae occur in documents and papers in a wide variety of styles. However, two major classes can be distinguished — formulae that occur in-line, such as $\sin^2 \theta + \cos^2 \theta = 1$, and formulae that occur out-of-line, such as

$$\binom{p}{2} x^2 y^{p-2} - \frac{1}{1-x} \frac{1}{1-x^2} \quad (4-1)$$

Depending on the class of the formula, different rules need to be applied to determine the choice of fonts and the placing of characters. These rules are subtly different according to whether the formula is in-line or out-of-line, and also whether the text being formatted is within a fraction, etc. For example, notice in equation 4-1 that the widths of the gaps around a minus sign differ according to whether the sign occurs in a superscript or not, and that the distance that superscripts are raised differs according to whether the subscript is in a denominator of a fraction or not. Not all the differences are as subtle as these, though. Fractions that occur in-line should be typeset as $\frac{a}{b+c}$, whereas if they occur out-of-line they should be typeset as

$$\frac{a}{b+c}$$

In accordance with T_EX the collection of rules that need to be applied is determined by the *style* of the formula; in-line formulae are in *text style* (S_TY_LE_T), and out-of-line formulae are in *display style* (S_TY_LE_D). Several other styles occur in formulae — *script style* (S_TY_LE_S) uses a smaller font size for superscripts, and *script script style* (S_TY_LE_{SS}) uses a still smaller font size in superscripts of superscripts. In addition there are four other styles that differ in certain finely tuned details.

In addition to choosing the fonts and positions of superscripts, good quality mathematical typesetting requires judicious selection of the amount of space between symbols; for example in

$$x + y = \max\{x, y\} + \min\{x, y\}$$

the spacings between the characters has been chosen according to the class of the symbols: whether they are operators, brackets, punctuation, etc. Furthermore note that the gap between the y and the close brace has been increased by the so-called *italic correction*. Without these subtle choices, the formula would have looked like

$$x+y = \max\{x,y\} + \min\{x,y\}$$

which is probably the best you can do with naive use of slug environments¹¹. The different classes of symbol

¹¹ Incidentally, the two examples have been produced as follows:

```
@maths[x+y=max{x,y}+min{x,y}]
```

and

```
@centre[@i(x)+@i(y) = max{@i(x),@i(y)}+min{@i(x),@i(y)}]
```

showing that you do not always have to work hard to get good quality output.

are ordinary (corresponding to variables), operators (such as Σ), binary operators (such as + and -), relational operators (such as = and <), open brackets, close brackets, and punctuation. You will occasionally need to choose a class for a new symbol that you want to introduce into a formula; if so, read Knuth's description [T_EX, Chapter 13].

Finally, mathematical formulae frequently require large parentheses to be constructed. For example in the case of the matrix

$$L = \left\{ \begin{array}{cccc} 1 + x & \begin{array}{c} (20) \\ (02) \\ x + y \end{array} & 3.14159 & \frac{1}{a + \frac{1}{a+b}} \\ 3i & 4 + 5 + 6 & \sin x & \sin y \end{array} \right\}$$

the braces need to be constructed from simpler fragments.

An effective mathematical typesetting system will automatically decide on which rules to apply, without the need for the user to be aware of them. For example, the first four formulae in this section were obtained by typing

```
@m{@sup(sin,2) @g(q) + @sup(cos,2) @g(q) = 1}
@begin{maths, label TEX-p68}
  @paren[@atop(p,2)]@sup(x,2)@sup(y,p-2)-@fract[1,1-x]@fract[1,1-@sup(x,2)]
@end{maths}
@m{@fract(a,b+c)}
@maths{@fract(a,b+c)}
```

4.27.2 Simple formulae

To place a mathematical formula in line, use the `m` environment. This acts like a slug environment, but it invokes special processing of its body. Spaces and blank lines are ignored (with the exception noted below), and Mint automatically places the appropriate amount of space between symbols. For example, both `@m(x=y)` and `@m(x = y)` produce $x = y$. The only time you will need spaces will be to separate operators from variables in cases where there is otherwise an ambiguity. For example `@m(sinx)` produces $\sin x$, whereas `@m(sin x)` produces $\sin x$. Note that you do not need to specify that `sin` should be in the regular face, or that `x` should be in the italic face; Mint uses tables to find out this information. (These tables are loaded as a part of Mint's initialization. You can load other entries into the tables if you wish; see below.) Occasionally you will need to select a font within the `m` environment: this is done using slug environments in the normal way. For example `@m(sin @g(q))` produces $\sin \theta$. Only face codes and font sizes can be altered in this way — you cannot (for example) use `@+` or `@-`.

Always use the `m` environment for variables. Even though it is superficially like the `i` environment, it places the italic correction after the identifier, which makes the output much more pleasing. For example, `@i(j)` produces j ; and `@m(j)` produces j ; the first j is too close to the semicolon.

Mint will place the whole of the formula on one line; sometimes this may not produce pleasing layout. If you feel that better layout can be obtained by breaking the formula across two lines, you can indicate to Mint where the best place to do this is by using `cbreak`; this has no effect if the formula does not need to be broken¹².

A formula is displayed out of line using the `maths` box environment¹³. The `maths` box environment, apart from processing its contents using special formatting rules, is otherwise like any other box environment: it takes all the same box environment parameters (though not all of them are used), and it can be incorporated into other box environments (figures, tables, etc.) or placed into headings or footings, in just the same way as any other box environment.

$$T_{1,k} = J - \sum_{j=2}^{\infty} \frac{1}{3} \left(\frac{4}{2^{2j}} - 1 \right) a_j \left(\frac{b-a}{2^k} \right)^{2j}$$

This equation states that, if we perform the trapezoidal rule to approximate J using a spacing $h_1 = (b-a)/2^{k+1}$ and $h_2 = (b-a)/2^k$ then the resulting approximation has a leading term in the error of the order of h_2^4 . The approximation $T_{1,k}$ is, in fact, precisely the parabolic rule for 2^k subintervals¹⁴.

Figure 5. Example of a mathematical formula in a figure

If you want several formulae together in the same box, separate them by at least one blank line: the distances between the formulae will be the same as the current line gap.

4.27.3 More complex formulae

4.27.3.1 Formula types

To obtain formulae more complex than simple algebraic equations, it is necessary to tell Mint about the type of the formula. This is done using a statement of the form

```
@FormulaType (Param1, Param2, ..., Paramn)
```

Several formula types are built into Mint, which understands which formatting rules to apply to the whole formula, and to each of the parameters. These formula types are only understood within the in-line and out-of-line mathematical environments.

There are formula types to describe most of the commonly occurring situations, and others can be added

¹² Console yourself that it is easy to check your text using cross-proofing.

¹³ `math` is also allowed.

¹⁴ From Ralston, *A First Course in Numerical Analysis*. McGraw-Hill.

on request¹⁵. For example, to obtain an in-line fraction, you write `@m(@frac(a,b+c))`. Mathematical formula types can, of course, be nested arbitrarily, and the change of formatting rules for each formula type is made automatically. Currently the following formula types are defined.

`@sup(a,b)` This supscripts the first argument; for example a^b . The amount the superscript is raised is a function of the style of the formula.

`@sub(a,b)` This subscripts the first argument; for example a_b . The amount the subscript is lowered is a function of the style of the formula.

`@supb(a,b,c)` This both supscripts and subscripts; for example a_c^b .

`@frac(num,den)` This creates a fraction. The numerator and denominator are centred, and the divisor line is placed on the same horizontal line as a `-` sign. `@atop(a,b)` is similar to `@frac`, but there is no divisor line, and the formula rests on the reference line (***) .

`@sum(from,to,arg)` In in-line formulae the limits are placed after the sigma, in out-of-line formulae they are placed below and above the sigma. Either or both the limits can be omitted; you should do this by using `skip` in place of the parameter, e.g. `@sum(skip,skip,@sup(i,2))`. `@prod`, `@union` and `@inter` produce products, unions and intersections.

`@int(from,to,arg)` The limits are placed after the integral sign in both in-line and out-of-line formulae. Either or both of the limits can be omitted, by replacing them by `skip`.

`@paren(arg)` This encloses the argument in parentheses that are just larger than the argument. First several standard parenthesis characters are examined, to determine whether any of them are large enough; if not, Mint constructs parentheses from simpler fragments. For example, in

$$\left(\left(\left(\left(a\right)\right)\right)\right)$$

the outer x parentheses are constructed from fragments. In addition to parentheses, Mint allows brackets (`@bracket(arg)`), braces (`@brace(arg)`), diamond brackets (`@diamond(arg)`), floor symbols (`@floor(arg)`), ceiling symbols (`@ceiling(arg)`), single bars (`@bar(arg)`), and double bars (`@dbar(arg)`).

¹⁵ And after paying a suitable fee.

`@matrix(arg)` This allows matrices to be constructed. The argument must be either several `@mrows` or `@mcols`; for example `@matrix(@mrow(1,2),@mrow(3,4))` will construct a 2×2 matrix. There can be up to 8 elements in each of the rows and columns, and it is required that the rows (or columns) have the same number of elements. Mint will create a matrix in which each element is centred horizontally and vertically, and with the spacing between the rows and columns equal to one quad. `@mrow` and `@mcol` should only be used within `@matrix`.

`@lbrace(arg)` This creates a left brace that is just bigger than the argument, as for example in

$$|x| = \begin{cases} x, & \text{if } x \geq 0; \\ -x, & \text{if } x < 0. \end{cases}$$

`Rbrace` places a right brace after its argument.

`@limop(op,subs,arg)` This is used for operators which have expressions placed beneath them, as in the case of

$$\max_{1 \leq n \leq m} \log_2 P_n \text{ and } \lim_{x \rightarrow 0} \frac{\sin x}{x} = 1$$

The `op` must be a single lexeme; otherwise there are no restrictions on it.

4.27.3.2 Labelled equations

Some mathematical formulae in documents require labels, whereas others may not require them. Mint provides a means of attaching labels to selected formulae through the use of counters; these labels have all the properties of other labels: they can be used to extract any counter value from the contour associated with them, and they can be converted using any style. To associate a label with a formula, include the optional parameter `label` in the environment parameters of the `maths` box environment. For example

```
@begin(maths, label = Newton-Raphson)
@sub(x,i+1) = @sub(x,i) - @frac{f(@sub(x,i)),f'(@sub(x,i))}
@end(maths)
```

will cause Mint to append an equation number to the end of the equation, as follows:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad (4-2)$$

The equation can then be referred to by the label. The macro `eqn` produces the equation number; however, the label can be referred to in any other way. For example

```
equation @eqn(Newton-Raphson) on page @pageno(Newton-Raphson) is the familiar
Newton-Raphson iteration formula
```

tells you that equation 4-2 on page 117 is the familiar Newton-Raphson iteration formula.

The equation number in the `maths` environment is centred vertically about the equation it labels {not yet}; if several formulae occur in a `maths` environment, only the last one is labelled.

4.27.4 Advanced concepts

In this section I describe several advanced features that allow fine tuning of formulae. Not all the notions expressed in this section are quite firmly grounded.

4.27.4.1 Mathematical fonts

When performing mathematical typesetting, Mint assumes that there are 10 fonts associated with each galley. These are fonts `m0`, `m1` and `m2`, in each of the sizes `n`, `s` and `ss`, and an extension font `mex`. Mint uses characters from these fonts to construct its output; it assumes that face code `m0` will contain regular characters to be used for operators like `sin`; that face code `m1` will contain italic characters to be used for variables like x ; that face code `m2` will contain special characters for operators, etc.; and that `mex` will contain a collection of special symbols for constructing large parentheses, etc. Font size `n` is used for `StyleT` and `StyleD`; font size `s` is used for `StyleS`; and font size `ss` is used for `StyleSS`.

With the exception of `mex`, these fonts are handled in the same way as any others — they are associated with the galley in the same way as other fonts, and they can have other characters, gaps or icons substituted for characters in them. `m0`, `m1` and `m2` are slug environments like `r`, `i`, etc. Note that the fonts `m0`, `m1`, `m2` and `mex` need to be associated with each galley that is to be used for mathematical formulae¹⁶.

The `mex` font is not like other fonts and a special set of statements is provided to manipulate it. Mint assumes that the font will contain a variety of characters drawn from other fonts, in character positions that allow it to construct large parentheses, etc. (It may also contain icons.) The sizes of the characters in it is determined by the font from which the character is borrowed, rather than by information contained in the `mex` font itself.

The statements to manipulate the font are: `CreateMFont`, which creates a new empty `mex` font; `AssocMFont`, which associates the font with some galley; and `SetMexChar`, which alters the parameters associated with a `mex` character. Normally you will never need to use these statements. They are described in section 4.9.5.

4.27.4.2 Changing fonts

Normally Mint will make the appropriate choice of font size and face code for formulae; these choices can be overridden by using slug environments within the `m` and `maths` environments. If you use a face code slug environment, this will override the automatic face code changes, but not the automatic font size

¹⁶ In particular, you cannot put formulae into footnotes and page headings and footings unless you have associated the fonts with the appropriate galley.

changes; similarly if you use a font size slug environment, this will override the automatic font size changes, but not the automatic face code changes. For example

```
@maths{@sub(a,1)+@fract(1,@sub(a,2)+@fract(1,@sub(a,3)+@fract(1,b)))}
```

produces

$$a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \frac{1}{b}}}$$

whereas

```
@maths{@n[ @sub(a,1)+@fract(1,@sub(a,2)+@fract(1,@sub(a,3)+@fract(1,b)))]}
```

produces

$$a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \frac{1}{b}}}$$

(Both rather nasty.)

If you use both font size and face code slug environments, Mint's automatic choice is completely overridden.

Spaces and other typographical layout characters become significant if you use a slug environment within `@m` or `@maths`. For example

`x + a constant greater than zero`

4.27.4.3 Defining symbols

Mint's spacing rules are applied by classifying each symbol that occurs in a formula, and then selecting the gaps needed between two symbols according to their classification. There are basically no in-built rules; instead symbols are placed in a table with their class, and this table is examined when needed. Several symbols are placed in the table when Mint starts; others can be added as needed. The statement

```
@MDef (Source, Destination, SymbolClass, FaceCode)
```

specifies that if `Source` occurs in the input, then it is to be replaced by `Destination` in face code `FaceCode`, and that for the purposes of determining the spacings, it is to be regarded as a `SymbolClass`. The spacing classes are `relop`, which is used for relational operators; `binop`, which is used for binary operators such as `+`; `op`, which is used for operators such as `sin`; `ord`, which is used for variables; `open`, which is used for left parentheses and other left delimiters; `close`, which is used similarly for right delimiters; and `punct`, which is used for punctuation. `<<Monop, Num>>` Symbols can also be defined as

`over`, `under` and `after`; these are described in the next section. For example, to cause Mint to recognize `Max` as an operator, to be output in `m0`, do

```
@MDef (max, max, op, m0)
```

and to allow `eqv` to be used in formulae such as `@m(x eqv b)`, do

```
@MDef (eqv, @char(#21), relop, m2)
```

when you will obtain $x \equiv b$.

If Mint cannot find a symbol in its table, it will assume that it is either a variable (and output it in face code `m1`), or a number (and output it in face code `m0`), in the current font size. These are the only assumptions that Mint makes which concern the maths fonts.

4.27.4.4 Inflected symbols

Symbols, such as variables and operators, can be *inflected*. If a symbol is inflected it has an accent above it, below it, or after it. For example, \tilde{x} , $\underset{\sim}{x}$, and x' are all inflected. Mint provides a general means of inflecting symbols, and allows several simultaneous inflections, as in $\tilde{\underset{\sim}{i}}$; up to seven accents can be associated with a symbol, though generally you will need only one.

To be able to use an accent it must first be defined. An accent belongs to a symbol class, in just the same way as an `operator`, `open` and `close` does; the classes are `above`, `below` and `after` (others will be added later), and `mdef` is used in the same way to define one. For example,

```
@MDef (Vec, ~, Under, M0)
@mDef (OTilde, ~, Over, M0)
@mDef (OBar, -, Over, M0)
```

specifies that `vec` will be an accent that will appear as a tilde in font `m0` under the inflected symbol. Also, `otilde` and `obar` will cause the corresponding accent over the symbol.

To inflect a symbol, it is simply followed by the accent; if there are several accents they will be placed above each other, below each other or after each other, in the order in which they appear. For example `@m(x obar odot)` produces $\dot{\bar{x}}$. Note that you don't need to be concerned about the positioning of the accent; Mint has enough information about the characters to place the accents in the correct place.

The following accents are defined in Mint. `OTilde`, `OBar`, `OHat`, `OVec` and `ODot` place accents over the symbol; `UTilde`, `UBar`, `UVec` and `UDot` place accents under the symbol; and `Tick` places a tick after the symbol. You may choose better identifiers if you wish. For example, the definition

```
@MDef (^, ^, Over, M0)
```

will allow you to write `@m(x^+y^)` to obtain $\hat{x} + \hat{y}$.

4.27.4.5 Grouping subformulae

Within the `m` and `maths` environments it is sometimes necessary to group together the symbols of a subformula: the `expr` environment performs this grouping action. This is frequently required when there is a comma in a formula: for example `@m(@sub(K,@expr(i,j)))` produces $K_{i,j}$; other examples occur when it is wished to cause some of the mathematical environment parameters described below to apply to a subset of the symbols in a formula.

4.27.4.6 Controlling the style

Sometimes Mint's automatic choice of styles is not appropriate. There are several ways that the choice can be overridden; for example, the use of font size slug environments has some of the effect of controlling the style, though as was seen in the continued fraction example, defining only the font size may not always produce pleasing results. Mint allows fine control over the *style*, however, without preventing automatic style changes from occurring subsequently.

If a style different from `StyleT` is needed for in-line formulae, the environments `md`, `ms` and `mss` (as well as `mt`, which is the same as `m`) can be used in place of `m`. For example, `@md(@frac(a,b))` produces $\frac{a}{b}$. If a style different from `StyleD` is needed for out-of-line formulae, the additional box environment parameter `style` can be used with the `maths` environment:

```
@begin(maths, style=stylet)
```

4.27.4.7 Mathematical environment parameters

The appearance of a formula that is within a formula type is determined in part by several *mathematical environment parameters*, which control the style and positions of symbols. Mathematical environment parameters are inherited from the parent formula type, and are modified in accordance with formatting rules built into the formula type, in a manner very similar to that for box environment parameters. The user can change the values that are inherited using similar techniques. For example, you can write

```
@begin(frac, style = stylet)
1, b+c
@end(frac)
```

inside both in-line and out-of-line formulae. The syntax for mathematical environment parameters is the same as that for box environment parameters — they may occur in any order; they are separated by commas; and the argument may be preceded by an equals.

The parameters are as follows.

Style This takes values from `StyleT`, `StyleD`, `StyleS` and `StyleSS`. The specified style is imposed on the formula type. See below for easier ways of using this style parameter.

Xposn This takes values from `Left`, `Centre`¹⁷ and `Right`. It specifies how the mathematical formula will be positioned inside the parent formula: `Left` flushes the formula to the left; `Right` flushes it to the right; and `Centre` centres it. This only has an effect in those environments where a subformula has this degree of freedom — in particular, this is so with fractions. See below for easier ways of using this style parameter.

Yposn This takes values from `Relative`, `Above`, `Centre` (`Center`) and `Below`. `Relative` causes the reference point of the subformula to line up with that of the parent formula; `Above` and `Below` cause the subformula to be positioned above and below the position of a `--` sign, and `Centre` centres the formula about this line. See below for easier ways of using this style parameter.

Xgap, Ygap These specify the gaps that will occur between rows and columns of matrices; they are specified in `quads`. Their default values are `1 quad`; the inner matrix in the example in section *** had `Xgap` and `Ygap` specified to be `0.5quad`.

Type This takes one of the values `relop`, `binop`, `op`, `ord`, `open`, `close`, `punct`, (and, because I can't easily prevent it, `over`, `under` and `after`, although these won't have any effect), and coerces the subformulae to be of the specified spacing type. The value of this is when you want a composite object, such as \xrightarrow{e} not to be regarded as an `ord`, which it normally would be. For example, assume that you want \xrightarrow{e} to be a `relop`; then you write

```
A @begin(col,type=relop) e, @vsp(-0.3em), rarrow @end(col) B
```

The effect is

$$A \xrightarrow{e} B$$

whereas if you simply write

```
A @begin(col) e, @vsp(-0.3em), rarrow @end(col) B
```

you get

$$A \xrightarrow{e} B$$

Some people are concerned about the difference. See below for easier ways of using this style parameter.

Since instances of `@begin(expr, style = s)`, `@begin(expr, type = t)`, `@begin(expr, xposn = x)` and `@begin(expr, yposn = y)` occur frequently, Mint provides abbreviations. In each of these cases you can write, for example, `@styles(...)`, `@ord(...)`, `@left(...)`, and `@below(...)`. In all cases the identifier you use is the same as the style, type, xposn or yposn you require; alas, since `centre` is both a `xposn` and `yposn` parameter, you must write `@xcentre(...)` or `@ycentre(...)` (or `xcenter`, `ycenter`). You can modify any of the maths environment parameters, using `@begin(...)`, in the usual way. (However, if you use write, for example, `@begin(stylelet, style=styled)`, you will get `styled`.)

¹⁷ also `Center`.

A comprehensive example of the use of these parameters is provided by the following continued fraction. First, I wanted to override the style changes associated with `@frac`, but I did not want to use the `@n` slug environment, since I wanted style changes to occur in the subscripts. Second, I wanted the numerators to be flushed to the left, rather than being centred. The box

```
@maths{@sub(a,1) +
  @begin(frac, style styled)
    @left(1), @sub(a,2) +
      @begin(frac, style styled)
        @left(1), @sub(a,3) +
          @begin(frac, style styled)
            1,b
          @end(frac)
        @end(frac)
      @end(frac)
    @end(frac)
  }
```

(I never said it was going to be elegant) produces

$$a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \frac{1}{b}}}$$

4.27.4.8 Tabular layout of formulae

Tabulations may be used in the `maths` environment to position several independent formulae on one line, and to position formulae on several lines one above another. The `maths` environment responds to tabulations in much the same way as the `align` environment; that is, `@t` lays down a tabulation, `@\` drags the lexemes to its left up to the next tabulation, and `@>` and `@<` centre the text between them around the next tabulation. They can only occur outside formula types, and each of them causes the current formula to end and a new one to start on the same line (I really should alter that some time). For example,

```
@maths{a+b+c@>d+e}
```

produces two formulae, and

```
@maths{@frac(a+b+@>c+d,x+y)}
```

is illegal.

The `maths` environment is defined to have `justifyleft` and `justifyleftlast` both `true`, `justifyright` and `justifyrightlast` both `false`, and to have its tabulations initially set up by `tabdivide 2`. Formulae are centred by default because the `maths` environment places a `@>` at the beginning of its body, and `@<` at the end; and the equation number is flushed right because the string which is injected into the `maths` body is terminated by `@\`. If you are controlling the layout yourself, you probably do not want the `@>` or the `@<`; to prevent them occurring, use the extra environment parameter `autotab`, which takes values `on` and `off`. For example

```

@begin(maths, tabdivide 9, autotab off, label Aho-Hopcroft-Ullman-p238)
@>\@>E = @bracket(@matrix(@mrow(1,0),@mrow(0,2)))@<
@>@r(and)@<
@>F = @bracket(@matrix(@mrow(0,0),@mrow(0,0)))@<@>\@>\@>
@end(maths)

```

produces

$$E = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \quad \text{and} \quad F = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad (4-3)$$

and

```

@begin(maths, tabclear, tabset 1.5in, tabset 2.75in, tabset 4.0in, autotab
off)
@>@sub(s,1)=@sub(a,21)+@sub(a,22)@>@sub(m,1)=@sub(s,2)@sub(s,6)
@>@sub(t,1)=@sub(m,1)+@sub(m,2)

@>@sub(s,2)=@sub(s,1)-@sub(a,11)@>@sub(m,2)=@sub(a,11)@sub(b,11)
@>@sub(t,2)=@sub(t,1)+@sub(m,4)

@>@sub(s,3)=@sub(a,11)-@sub(a,21)@>@sub(m,3)=@sub(a,21)@sub(b,21)

@>@sub(s,4)=@sub(a,12)-@sub(s,2)@>@sub(m,4)=@sub(s,3)@sub(s,7)
@end(maths)

```

produces

$$\begin{array}{lll}
s_1 = a_{21} + a_{22} & m_1 = s_2 s_6 & t_1 = m_1 + m_2 \\
s_2 = s_1 - a_{11} & m_2 = a_{11} b_{11} & t_2 = t_1 + m_4 \\
s_3 = a_{11} - a_{21} & m_3 = a_{21} b_{21} & \\
s_4 = a_{12} - s_2 & m_4 = s_3 s_7 &
\end{array}$$

In fact there is a subtle difference between @> and @\ in the maths environment. The tabulation @> places an empty separator before the tabulation, whereas the tabulation @\ doesn't. The effect of this is that the @> tabulation always moves the input that follows up to the next tabulation, or centres it about the tabulation, whereas the @\ tabulation may act as a normal tabulation, and shift all the lexemes that follow it out to the next tabulation. It turns out that the rules are difficult to remember, so I have prepared the following tables which show the effect of the various tabulations.

Input			
123@>456	123	456	
123@>456@<	123	456	
123@>456@>	123		456
123@>456@>		123456	
123@>456@<		123456	
123@>456@>		123	456

(Beware: in appendices the equation numbering will appear like A-5, for example; however, the counter `EquationNo` is not reset to 1 at the start of each appendix. I should fix it, but it's messy; on the other hand, if you have read so far, you should know how to reset the counter back to 1 yourself.)

4.27.5 Really advanced features

The features that have been described above should allow most Minters to produce high quality mathematical output. However, it seems to be a characteristic of those who wish to produce really high quality output that they are not satisfied until they understand all the inner workings of the tools they are using. It is for these people, and also incidentally for those for whom the above facilities are not sufficient, that this section has been written. Unless you are a fanatic, you should stop reading this now.

Well, so you are a fanatic. Several of the features I am about to describe are repetitions, in more detail, of what has been said before. In addition I will describe two other mathematical environments and some more environment parameters.

4.27.5.1 Mathematical layout vectors

Every object in the mathematical environments in Mint is characterized by three vectors. These vectors, together with the font and characters making up the object, are sufficient to describe any formula that is laid out using Mint. Understanding how the values of the vectors are computed should, then, be sufficient to understand how Mint will treat each formula.

Each object (single character, or sequence of characters making up a single lexeme, such as \sin or $+$), is assumed to sit within a rectangular *bounding box* and to have an *origin*, which is used for determining how to place characters next to each other. The position of the bounding box from the origin is described by two of the three vectors: $(X0, Y0)$, the vector to the bottom left corner of the bounding box, and (XX, YY) , the vector to the top right hand corner of the bounding box. These vectors are *intrinsic* to the symbol; that is, they are independent of the context in which the symbol occurs. (Just which symbol corresponds to a particular sequence of characters in the manuscript file is determined by the *style* — part of the context — in which the symbol occurs.) The third vector, $(XRe1, YRe1)$, determines the position of the origin relative to the origin of the enclosing parent mathematical environment. Each environment applies different rules for determining this last vector; these rules describe, to a large extent, the differences between the different mathematical environments. One important characteristic of this description is the *placing freedom* — the freedom to shift the object in the X or Y plane according to environment specifications given by the user. Two classes of freedom are recognized by Mint: *weak* freedom, which can be overruled by Mint's internal rules, and *strong* freedom, which Mint always responds to. An example of weak freedom, which is overruled, occurs in the formula

```
@m{A @begin(expr, xposn=left) B @end(expr) C}
```

since in this case B will simply be placed in the same position that it would be if there were no specification of the $xposn$. An example of a strong freedom occurs in the formula

$$\@m\{A @begin(expr, xposn=3.5quads) B @end(expr) C\}$$

since in this case the origin of B will be placed 3.5 quads from the origin of the parent environment. All freedoms specified in section 4.27.4.7 are weak; those described below are strong.

Even in this really advanced section, it isn't appropriate to give all the details of the computations Mint performs. Instead I will summarize them, omitting details of how italic corrections are incorporated.

Expr	And m , etc. The bounding box of the first object is set flush with the bounding box of the parent; subsequent objects are placed such that the X-part of the origins are coincident with the right-hand edge of the bounding box of the preceding object; Y-freedoms are honoured, though if not otherwise specified the Y-part of the origins lie on the same horizontal line as the Y-part of the origin of the bounding box of the parent. The size of the bounding box of the parent is such that it just encloses all the symbols within it. There are, however, details of italic correction calculation which have been omitted from this description.
Fract	Etc, etc.
Sum	And Prod, Inter, Union. Etc.
Sup	And Sub, Supb. Etc.
Paren	Etc.
Matrix	Lots of goodies here.
Row	This environment is superficially similar to the <code>expr</code> environment and to the <code>mrow</code> environment. It takes up to eight subformulae, and places them next to each other. Each subformula can be strongly shifted in the X-plane, and weakly and strongly shifted in the Y-plane, and are otherwise placed flush with each other in the X-plane, and with the Y origins on the same horizontal line. A subformula that has its X-position specified will push to the right all the subformulae that are on its right.
Col	This environment is similar to the <code>row</code> environment. It places formulae above each other, such that the Y-origin of one box sits on the top of the bounding box of the subformula beneath it, and with the boxes centred in the X-direction. The environment responds to both weak and strong shifts in the X-plane, and strong shifts in the Y-plane.

4.27.5.2 Styles

The style of a formula determines various appearances of the formula, for example the fonts to use, the positions of limits in summations, and the positions of superscripts and subscripts. The description given by Knuth is complete.

4.27.5.3 Types

The type of a symbol determines the space that is placed between the symbol and its neighbours. A two-dimensional array is needed to describe the spacings; in addition, the spacings are a function of the current style. The description by Knuth is barely adequate. To obtain high quality output it has been necessary to make many modifications to the spacings, and it has been necessary to introduce two new types. These are: `Num`, the type of a number, which allows a different space to be placed between the 2 and the x in $2x$ than between the y and x in yx (note how this differs from x^2); and `flush`, which causes no space to be placed between such a symbol and its neighbours (for example, `@m(x;)` produces $x;$, and `@m(@flush(x);)` produces $x;$). All spacings are of type `Flush`.

4.27.5.4 Spacings, etc.

Fanatics frequently have need to control precisely the layout of a formula. Mint provides several facilities for this: spacings; the `row` and `col` environments; and positioning parameters. These are all described in this section.

You can put an arbitrary amount of space between two symbols using `vsp` and `hsp`. These effectively generate empty lexemes of zero width and the specified height, and of zero height and the specified width, respectively. In both cases the symbols belong to the type `flush`, so there is no extra space added by Mint. As in other contexts, the parameters to `vsp` and `hsp` can be absolute units, font-relative or page-relative. For example, assume you want precisely a quarter of an inch of space between A and the equals symbol in a formula. You write `@m(A @hsp(0.25 inches) = B)` and get $A = B$.

The `row` and `col` environments play a similar role in the mathematical environments as do the `describe` and `multiple` environments when manipulating boxes; that is, they allow you to create composite objects in which inner objects are placed side by side, or stacked on top of each other. `Row` places objects side by side; each of its parameters is placed flush with each other. For example

```
@maths(@row (A, +, B))
```

produces

$$A+B$$

Note the difference between this and

```
@maths(A + B)
```

which produces

$$A + B$$

The `col` environment places objects on top of each other; there is an example in section *** above. Note that the parameter to `hsp` and `vsp` can be negative.

The `xposn` and `yposn` environment parameters take, in addition to the values specified in section 4.27.4.7, both explicit values, and *label values*. An explicit value is given as a value of some length, for example `3 inches` or `4ems`. These specify the distance the object will be placed from the origin of the parents bounding box. A label value is used for aligning equations within a single mathematical environment, or across several environments. In order to use a label, it must first be declared — either in the same `m` or `maths` environment as it is being used, or in some previous environment. The environment parameter `label` declares the label; for example

```
@begin(expr, label here) a + b @end(expr)
```

defines the label `here`. Labels can be used as values for the `xposn` and `yposn` environment parameters; for example

```
@begin(expr, xposn here) p + q @end(expr)
```

will strongly set the `xposn` of the second `expr` to the same value (relative to the origin of the enclosing box) as the first `expr`.

4.27.5.5 Mathematical font parameters

There are several parameters that control the spacings between the components of mathematical formulae; these have been carefully chosen by your implementer to give the most pleasing appearance to formulae. You can alter some of them, using the statement `MathsParams`, but my advice is to leave them alone, unless you find you really need to do so. The statement

```
@MathsParams(F1 = 0.30ems, F2 = 0.07ems, F3 = 0.05ems,  
             P1 = 0.17ems, P2 = 0.14ems, P3 = 0.09ems,  
             B1 = -0.09ems, B2 = -0.07ems, B3 = -0.07ems, B4 = -0.08ems,  
             E1 = 0.04ems)
```

will set the values of those parameters that are accessible; there are many other parameters that can only be set by taking apart Mint. The choice of which can be altered and which cannot has been determined in part by reading the `TEX` book; however, I regard Knuth's choice as fairly arbitrary, and I have not hesitated to choose other values for parameters where I feel that more pleasing output can be achieved. The parameters that can be altered have the following effect.

- | | |
|-----------|--|
| F1 | The height of the divisor line in a fraction above the base line. |
| F2 | The height of the bottom of the numerator of a fraction above the divisor line. Twice this value is used to separate the top of the denominator from the divisor line. |
| F3 | The amount that the divisor line is shorter at the left than the larger of numerator and denominator. |
| P1 | The extra height of a superscript in <code>DMode</code> above the position it would have with style parameter <code>YPosn</code> equal to <code>Below</code> . |

- P2** The extra height of a superscript in `TMode`, `SMode` and `SSMode` above the position it would have with style parameter `YPosn` equal to `Below`.
- P3** The extra height of a superscript in numerators and denominators above the position it would have with style parameter `YPosn` equal to `Below`.
- B1** The extra height of a subscript in `DMode` above the position it would have with style parameter `YPosn` equal to `Above`.
- B2** The extra height of a subscript in `TMode`, `SMode` and `SSMode` above the position it would have with style parameter `YPosn` equal to `Above`.
- B3** The extra height of a subscript in numerators and denominators above the position it would have with style parameter `YPosn` equal to `Above`.
- B4** An additional height to add to subscript positions in the case that there is a superscript present. (Note that all the `B` parameters are negative.)
- E1** The size of the gap to leave between a superscript or subscript and the expression being scripted. This gap is in addition to any italic correction that may be applied.

The values shown in the `MathsParams` statement are the values that the parameters take by default.

4.28 DP and Plot

<<Information on how to incorporate DP and Plot output into Mint will be given later. The following has proved to be of use to those who already have used DP and Plot.

Mint provides two environments, `DP` and `Plot`, that have their interpreters set so that they accept and interpret input appropriate for `DP` and `Plot`.

The input to the `DP` environment should be a file as produced by `DP`; it is normally included as follows

```
@begin(DP, width = 3in)
@Include (Mouse.Dp)
@end(DP)
```

Mint will perform the appropriate scaling, both for the target device and for the viewing device, should cross proofing be requested.

The input to the `Plot` environment should be a file as produced by `Plot`, with the device specified as `generic`. Mint will perform the appropriate scaling, as in the case of `DP`, but the quality of the output is much more dependent on having the resolution and size close to that finally needed.>>

4.29 Errors

There are four classes of error: *Warnings*, *Errors*, *Heresies* and *Fatal Errors*, in increasing severity. Warnings are given if Mint detects suspicious input that is not otherwise incorrect and that can be formatted appropriately; Error messages are given if the input cannot be formatted, but where it is possible to continue formatting the rest of the document. Heresies generally indicate problems inside Mint, where there is doubt about its ability to continue; and Fatal Errors indicate serious problems that prevent Mint from continuing. After a Warning or Error Mint continues; after a Heresy or Fatal Error it halts. It may be resumed, but the effects are unpredictable. You should report heresies and fatal errors to me using the report mechanism described in section 1.3.1.

Error messages appear in the lower window on the Perq screen, and are written off to a file with extension `.ERROR`. The message gives the input file location, and the reason for the error. If Mint hasn't yet been able to read any input, the part of the file name before the dot will be empty (thus you will find the errors listed in a file named `.error`). There is no complete list of error messages available, but they should be self-explanatory.

4.30 Quirks and Oddities

Mint has only just emerged from its shell into the light of day, so it still needs shaking down. In general, I have found it reasonably bug-free for my particular style of `.Mss` file, but others who have a different style have unearthed bugs. In addition it is not yet very robust, and the error messages it gives are sometimes misleading. The main problems are as follows — they will be fixed as soon as possible.

- The `multiple` environment is not like Scribe's. It is a fully-fledged environment, in which nest other environments; thus it can be used with the `describe` environments for the architectural design of box layouts. When used with `itemize` and `enumerate` it works as expected; with `description` it reveals all too clearly that `description` is a nasty hack, which should be replaced by `describe`.
- The macrogenerator has on occasion been found to be leaking an internal marker. The only effect of this is to produce on the Dover cover sheet a note that an illegal character has been detected, but the output should otherwise be correct. I know where it is coming from, but the macrogenerator is the least-loved module in the whole of Mint, and I hate delving into it. **Later note:** I think this is now fixed.
- The clipping algorithm for box borders clips to page boundaries, not page area boundaries, so that a box which gets split between two pages and which has a visible border will appear very strange. Since you probably didn't want the box split over two pages anyway, you could regard the appearance of the page as a quaint Mint warning message.
- Comments are handled by the macrogenerator, not the rest of the system. Since the macrogenerator's command conventions differ from those of the rest of the system (`=` is mandatory, and `@begin(x)` is not permitted) you have to be careful when commenting out text.

- Because there is no bullet in the Perq fonts, `itemize` environments are stripped of this necessary adornment.
- There seems to be a problem with the `@zsp` environment; or rather it has well defined formal properties, but these are not the properties expected by the unprepared user. Always place several newlines around the command, and then the new page may occur where you expect it. The definition of the `NewPage` macro has been fixed so that it adds the necessary newlines.
- Because the `W` statement rescans its argument, surprising results sometimes occur.
- The `maths` environment seems to work satisfactorily if it is fed the correct input; feed it incorrect input and it will corrupt the rest of Mint. If you use this environment and you get an unusual error message, it's worth while looking at your `maths` input carefully.
- The macrogenerator is a whole can of worms.