

CARNEGIE-MELLON UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE
SPICE PROJECT

Mint
Reference Manual

Peter Hibbard

14 February 84

Abstract

This document describes version 2B(12) 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.

Spice Document S153

Keywords and index categories: none

Location of machine-readable file: refman

Copyright © MCMLXXXIV 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 Notes about this version	1
1.1 Changes since the last release	1
1.1.1 Syntactic changes	1
1.1.2 Units of measurement	2
1.1.3 Galley properties	2
1.1.4 Fonts	3
1.1.5 Macrogenerator	3
1.1.6 Presentations	4
1.1.7 Tables of contents	4
1.1.8 Libraries	4
1.1.9 State files	5
1.1.10 Separate formatting	5
1.1.11 Annotations	5
1.2 Quirks and Oddities	6
2 Basic properties of documents	9
2.1 Input conventions	9
2.2 Document syntax	11
2.2.1 Syntactic metalanguage	11
2.2.2 Pseudo-Syntactic properties	12
2.2.3 Standard environments	12
2.2.4 Production rules common across several document types	16
2.2.4.1 Document environment syntax	17
2.2.4.2 Terminal environment syntax	17
2.2.4.3 Heading environment syntax	17
2.2.4.4 Section environment syntax	17
2.2.4.5 Item environment syntax	18
2.2.4.6 Title page environment syntax	18
2.2.4.7 Galley environment syntax	18
2.2.4.8 Page environment syntax	18
2.2.5 Document Syntax	19
2.2.6 Altering the syntax	19
2.3 Units of length	20
2.3.1 Absolute units	21
2.3.2 Internal units	21
2.3.3 Relative lengths	21
2.3.4 Scaling	22
2.3.5 Modifying environment parameters	22
2.4 Errors	23
3 Galleys	25

3.1 Galley components	25
3.1.1 Procedure families	26
3.1.2 Font families	27
3.1.3 Styles	28
3.1.4 Dominating environments	28
3.1.5 Installing a galley	28
3.1.6 Changing galley properties	29
3.2 Standard Galley Properties	30
3.2.1 Procedure families	30
3.2.1.1 Procedure family MainPF0	30
3.2.1.2 Procedure family MainPF1	31
3.2.1.3 Procedure family MainPF2	32
3.2.1.4 Procedure family FootPF0	33
3.2.1.5 Procedure family ContPF0	33
3.2.1.6 Procedure family OddsPF0	34
3.2.2 Font families	35
3.2.3 Standard styles	37
3.2.4 The galley parameters for the document types	38
3.2.4.1 Text, form 0	38
3.2.4.2 Text, form 1	38
3.2.4.3 Report, form 0	39
3.2.4.4 Article, form 0	39
3.2.4.5 Thesis, form 0	39
3.2.4.6 Slides, form 0	39
3.2.4.7 Manual, form 0	39
3.2.4.8 Manual, form 1	40
4 Boxes and Slugs	41
4.1 Box Environment Parameters	41
4.1.1 Standard attributes	42
4.1.2 Additional parameters	45
4.1.2.1 Style parameters for document types	45
4.1.2.2 Style parameters for other environments	46
4.1.2.3 Additional parameters for title pages	47
4.1.2.4 Itemize and Enumerate	47
4.1.2.5 Maths	48
4.2 The standard values for the environment values	48
4.3 Box procedures	55
4.4 Computations	57
4.4.1 Standard computations	57
4.4.2 Arbitrary computations	58
4.5 Miscellaneous layout statements	59
4.5.1 Spacing statements	59
4.5.2 Tabulations	60
4.5.3 The Align environment	60
4.5.4 The Describe environment	61

4.6 Slug Environments	63
4.6.1 Face Codes	63
4.6.2 Font Sizes	64
4.6.3 User Face Codes	65
4.6.4 Underlines, Overlines and Eraselines	65
4.6.5 Raster Functions	66
4.6.6 Scripting	66
4.6.7 Overprinting	67
5 Fonts	69
5.1 Font representations	69
5.2 Font families	69
5.2.1 Creating font families	70
5.2.2 Associating fonts with a font family	70
5.3 Logical fonts	71
5.3.1 Changing font mappings	71
5.3.2 Icons	72
5.3.3 New fonts	73
5.4 Character information	74
5.4.1 The values describing a glyph	74
5.4.2 Applying character information	76
5.4.3 Extracting character information	77
5.5 Spacing adjustments	77
5.5.1 Italic corrections	78
5.5.2 Kerning	78
5.5.3 Optical adjustments	79
6 Macrogenerator	81
6.1 Macro expansion	81
6.1.1 Input conventions	81
6.1.2 Defining macros	83
6.1.2.1 Accessing parameters	83
6.1.2.2 Accessing system values	84
6.1.3 Deferred Macros	84
6.2 Standard Macrogenerator Facilities	85
6.2.1 Special Macros	85
6.2.2 Extra macros	86
6.2.3 Summary of other macros	87
6.2.4 System attributes accessed via @Value	89
7 Cross references	91
7.1 Counters	91
7.1.1 Overview of Counters	92
7.1.2 Counter manipulations	92
7.2 Labels	93

7.2.1 Referring to labels	94
7.2.2 Undefined labels	94
7.3 Conversions	95
7.4 Standard Conversions and Counters	96
7.4.1 Conversions	96
7.4.2 Pseudo-counters	97
7.4.3 Non-basic conversions	97
7.4.4 Counters	98
7.4.4.1 Counters common to all document types	98
7.4.4.2 Counters in document types with footnotes and annotations	98
7.4.4.3 Counters in document types that have chapters	98
7.4.4.4 Counters in document types that have sections	98
7.5 Prefixes and postfixes	99
7.5.1 Standard prefixes	99
8 Page layout	101
8.1 Presentations	101
8.1.1 The structure of a presentation	101
8.1.2 Page styles	103
8.1.3 Layout routines	103
8.1.4 Defining new presentations	104
8.1.5 Making representations	104
8.1.6 Printing a presentation	105
8.2 Layout routines	105
8.2.1 Sorting the slugs and boxes	106
8.2.2 Page areas	106
8.2.2.1 Area parameter objects	106
8.2.2.2 Standard values of the page area parameters	107
8.2.3 Creating layouts	108
8.2.4 Actions of the layout procedures	109
8.2.4.1 The <code>Default</code> layout routine	109
8.2.4.2 The <code>TitlePage</code> layout routine	109
8.2.4.3 The <code>Contents</code> layout routine	109
8.2.4.4 The <code>Pasteup</code> layout routine	110
8.3 Standard presentations and printing	110
8.3.1 Page styles	110
8.3.2 Standard presentations	110
8.3.3 Printing the standard presentation	111
8.4 Page commands	111
8.4.1 Page headings and footings	111
8.4.2 Page offsets	112
8.4.3 Page skips	112
8.4.4 Example of headings and footings	113
9 Documentation aids	115
9.1 Bibliographies	115

9.1.1 Defining bibliographies	115
9.1.2 Citation collections	116
9.1.3 Citations	116
9.1.4 Causing the bibliography to appear	117
9.1.5 The standard bibliographies	117
9.2 Indexes	118
9.2.1 Index collections	118
9.2.2 Index entries	119
9.2.3 Causing the index to appear	119
9.2.3.1 The <code>Style1</code> indexing routine	120
9.3 Tables of contents	122
9.3.1 Declaring tables of contents	122
9.3.2 Generating the table of contents	123
10 Decorations	125
10.1 Borders and Border Styles	125
10.1.1 Border Styles	126
10.1.1.1 Lines	126
10.1.1.2 Patterns	126
10.1.1.3 Border Styles	126
10.2 Colours	127
10.2.1 Defining colours	127
10.2.2 Associating colours with objects	128
10.2.2.1 Associating colours with page areas	128
10.2.2.2 Associating colours with boxes	129
10.2.2.3 Associating colours with borders	129
10.2.2.4 Associating colours with characters and lines	129
10.2.3 The order of overlaying	130
11 Devices	131
11.1 Device definitions	131
11.1.0.1 Device drivers	131
11.1.0.2 Font formats	132
11.1.1 Device classes	132
11.1.2 Devices	133
11.2 Cross proofing	133
12 Mathematical Typesetting	135
12.1 Mathematical Typesetting	135
12.1.1 Basic Concepts	136
12.1.2 Simple formulae	138
12.1.3 More complex formulae	139
12.1.3.1 Formula types	139
12.1.3.2 Labelled equations	141
12.2 Advanced concepts	141

12.2.1 Mathematical fonts	142
12.2.1.1 Changing fonts	142
12.2.2 Defining symbols	143
12.2.2.1 Inflected symbols	144
12.2.2.2 Replacement text	145
12.2.3 Grouping subformulae	145
12.2.4 Controlling the style	146
12.2.5 Mathematical environment parameters	146
12.2.6 Tabular layout of formulae	148
12.2.7 Equation counters	150
12.3 Really advanced features	151
12.3.1 Non-fanatics stop here	151
12.3.2 Mathematical layout vectors	151
12.3.3 Styles	153
12.3.4 Types	154
12.3.5 Spacings, etc.	154
12.3.5.1 Spacings	154
12.3.5.2 Rows and columns	154
12.3.5.3 Positioning parameters	155
12.3.6 Mathematical font parameters	155
13 Alternative interpreters	157
13.1 Line drawings	157
13.1.1 DP and Plot	157
14 State Files and Libraries	159
14.1 Definition files	159
14.1.1 Standard state files	159
14.1.2 Creating standard state files	160
14.1.3 Low-level state file manipulations	161
14.1.4 Using definitions files	161
14.2 Libraries	162
14.3 Defining state	165
14.3.1 Basic definitions	165
14.3.1.1 The SpecialForm statement	166
14.3.1.2 The lexeme tables	166
14.3.1.3 Units	167
14.3.2 Syntax definitions	167
14.3.2.1 Syntax classes	168
14.3.2.2 Production rules	168
14.3.3 Galley definitions	169
14.3.3.1 Styles objects	169
14.3.3.2 Font families	169
14.3.3.3 Procedure families	170
14.3.3.4 Declaring galleys	170
14.3.3.5 Miscellaneous properties	171

14.3.4 Font definitions	171
14.3.4.1 Characteristics vectors	172
14.3.4.2 Font value objects	173
14.3.4.3 Font information	173
14.3.5 Counter and conversion definitions	175
14.3.5.1 Registering conversions and counters	175
14.3.5.2 Declaring counters	176
14.3.5.3 Associating conversions	176
14.3.5.4 Prefixes	177
14.3.5.5 Place conversions	178
14.3.5.6 Cross reference	178
14.3.6 Presentation definitions	179
14.3.6.1 Page styles	179
14.3.6.2 Presentations	179
14.3.6.3 Layouts	180
14.3.7 Bibliographies, contents definitions and indexes	180
14.3.8 Device definitions	182
14.3.9 Mathematical definitions	183
14.3.9.1 Composite symbols	183
14.3.9.2 Spacings	183
14.3.9.3 Standard symbols	184
14.3.9.4 Mathematical fonts	185
14.3.10 State file definitions	185

Part One

Notes about this version

This section of the Reference Manual describes the principal changes that have occurred in Mint since the last major release in November 1983, and describes some of the problems that still remain.

1.1 Changes since the last release

Mint has been extensively changed since the last major release; however most of the changes have been in internal organisation and in features that are not used by most users. The following summarises the changes; the reference at the beginning of each section points to the part of the Reference Manual that has more details.

1.1.1 Syntactic changes

[2.2] The opportunity was taken, with the introduction of state files (part 14), to revise the form of the parser and its tables. Mint has now very little in-built understanding of any environment, and what remains will be removed in a later version. This revision will be invisible to users.

- Mint now uses *syntactic classes* to describe the syntax, rather than *nonterminals*. This allows many of the restrictions in the old syntax to be lifted, and provides a simple means of defining new environments. The principal difference users will notice is that the syntax is more liberal, although the description is more complex.
- The statements **addrule** and **remrule** have been renamed to **addclass** and **remclass**, in order to reflect the change in the syntactic structure of documents. It is unlikely that users will need to use either of these statements.
- **Make** is no longer equivalent to **begin**; instead it is a statement that directs Mint how to set up the environment in which the document is processed. I have tried to make its effect be similar to that of the old **make** command, so in general you should not need to change your document. However, you should note that **make** now takes four parameters — **document**, the type of the

document; `form`, the form of the document; `device`, the device the document is intended for; and `rest`, a collection of parameters that get passed to the `begin` command for the document. Thus, the statement

```
@Make(Document=Text,Form=1,Device=Dover,Rest=@'{Indent=0in,Gap=3points})
```

corresponds to

```
@Make(Text, Form 1, Indent 0in, Gap 3points)
```

in the old system. The equals symbols are mandatory. This is because the `make` statement is interpreted by the macrogenerator. You can omit the `rest` parameter, the `device` parameter and the `form` parameter (for example you can write `@make(thesis)`). If your document does not start with a `make` statement, `@make(document=text,form=0,device=perq)` is assumed, as in the previous version. Be warned that there are a few flakey features lingering out there in the `make` statement, so follow the pattern I have given when you use it.

- The formal attribute `Need` now takes a value `all` as well as a vertical distance. This specifies that all the environment should be placed on the same page.

- The formal attribute `TabSet` now takes a list of horizontal distances:

```
@Begin(Describe, TabClear, TabSet 1in, 2ins, 3ins, 4ins)
```

1.1.2 Units of measurement

[2.3] A number of changes have been made in the way in which Mint handles units. These changes have been made to increase Mint's consistency, and to reflect a new treatment of devices and fonts.

- The major change that users will notice is that the misnamed `raster` unit has now been renamed to `iu` (internal unit).
- Mint now allows documents that are intended for photoreduction or magnification to be scaled, so that units refer to the final photocopied document, and not to the immediate output of the target device.
- The font-relative measures `em` and `quad` may be used in both horizontal and vertical directions. In the horizontal direction they measure the width of the letter 'M'; in the vertical direction they measure the vertical height above the baseline of the bounding box of the font. These two values need not be the same. This non-traditional treatment proves to be useful when describing characters (section 5.4).

1.1.3 Galley properties

[3] Based on experience gained with the previous version, I have been able to make Mint more comprehensively object-oriented, with statements that allow several types of object to be created and manipulated. Two areas have benefited from this approach — galleys, described here, and presentations, described in section 1.1.6.

- Procedure families and font families are now objects that can be manipulated within Mint. They can be declared, procedures and fonts can be associated with them, and they can be associated with galleys by means of statements in the .Mss file. The advantage is that the same procedure family and font family can be associated with several galleys, thus allowing a consistent behaviour of several related galleys, and allowing a wholesale change of fonts and procedures to be made to a galley at any time during processing the document. The principal effect users will observe is in the statements `assocproc` and `assocfont`, which now refer to procedure families and font families.
- There are several changes associated with creating galleys. Since these are not normally created by users, the differences will not be described here.
- Prefixes are now a part of the procedure family, rather than the environment, so allowing different galleys to have different prefixes for the same environment. Users are unlikely to make direct use of prefixes.

1.1.4 Fonts

[5] Major changes have occurred in Mint's treatment of fonts. There are three classes of change.

- Mint now allows sequences of characters to be replaced by a character, gap or icon. This allows Mint to use ligatures (for example, the sequence of characters f and i can be replaced by a single character fi), and provides a way of introducing accented characters (for example the sequence of characters a and " can be replaced by the single character ä).
- Mint now performs *kerning*, which is a second order adjustment of the position of characters. For example the two characters o followed by x need to be made closer together than they would otherwise be if they were simply placed adjacent to each other. The amount that character pairs need to be kerned is a property of the font design.
- Mint can now use a very complete description of the appearance of each character, to allow it to do high-quality typesetting. The information is not mandatory — Mint will make reasonable guesses about the values it needs, but mathematical typesetting does not make sense without some of the information.

1.1.5 Macrogenerator

[6] The days of the macrogenerator are numbered. Nonetheless it has managed to survive into this version, held together with string and bent pipe-cleaners. Most of the changes are patches to fix problems that have arisen in macroexpansion.

- Macro parameters are evaluated at most once, on being parcelled up prior to a macro call. The only time a string will be rescanned is when it is the body of a macro definition. A macro `eval` is provided to force re-evaluation.

- A macro argument is expected to be syntactically well-formed. It is syntactically well-formed if every embedded macro call is well-formed, by having matching brackets. For example

```
@mymacro(@yourmacro[argument)and a bit more]to finish off)
```

will be scanned as

```
yourmacro called with argument)and a bit more  
and  
mymacro called with <result of previous call>to finish off
```

- Quoted strings (for example @""quoted string") are also expected to be well-formed, so that

```
@""This is a quoted string @"(with a " in it)"
```

is all one quoted string.

- There are many more statements. They are described in section 6.2.

1.1.6 Presentations

[8] Presentations now make use of *page styles*, which are environment parameters, and *layouts*, which are collections of rules describing how to lay out pages. The correspondence between page styles and layouts is established by the user for each different presentation, so allowing fine control over the appearance of the document. Presentations, page styles and layouts are objects that can be declared in Mint; only *layout routines*, which take a layout specification and some set of galleys, are built in to Mint.

Presentations are not normally manipulated by users, so no further details will be given here.

1.1.7 Tables of contents

[9.3] Some document types now have a table of contents galley associated with them. When this is the case, Mint automatically generates a table of contents, though it is the user's responsibility to supply definitions of several macros to lay out the entries. A useful default set is provided in the standard release of Mint.

1.1.8 Libraries

[14.2] A number of collections of Mint statements are available in *library files*, which are ASCII files that have the extension .Lib. The statement `library` is a convenient way of including them:

```
@Library (TimesRomanKern)
```

1.1.9 State files

[14.3] The initial internal state of Mint is now read from a *state file* which is a binary file that loads up tables of definitions. Virtually all the internal state needed for a document can be stored in a state file — macro definitions, borders and border styles, `mod ifys` and `def ines`, the effects of the use of `assocfont` and substitutions, etc. A state file therefore provides a convenient closed-form encapsulation of a document design, and is a useful adjunct to a `.mss` file. A state file is created by Mint itself from normal `.mss` files; new state files can be created by incremental addition to a state file that already exists. Along with every release of Mint comes the set of source files necessary to construct the lowest level state files, those that correspond to the primitive document types (text, form 0; manual, form 1; etc). These files all have the extension `.Mint`; they will be referred to as “the `.Mint` files”.

Mint provides a version control mechanism as a part of the state file facility; however, it relies on the operating system’s notions of version control to implement it, so that any particular implementation of Mint may be deficient.

Because the state is constructed from Mint statements, it is misleading to talk about the “standard properties of Mint” — there is far greater flexibility in programming Mint than before. However, to avoid confusion the rest of this document describes the operation of Mint when it is fed the state files created from the statements that are part of the general release; those interested in programming different document designs should read section 14.3, and study the `.Mint` files.

1.1.10 Separate formatting

[1.1.10] The latest version of Mint permits documents to be broken into parts that can then be formatted separately. Mint saves the context that it uses for formatting each part, so that page numbers, section numbers, etc., are correct. The feature is not described yet in the documentation, but I will explain its use individually for those who need it.

1.1.11 Annotations

[1.1.11] Facilities now exist for including the editorial annotations of several editors in a document, and for these annotations to be viewed selectively. I have not yet included all the apparatus this facility needs in order to be used smoothly, so I will not describe it here. I am willing to explain it individually to anyone who needs it.

1.2 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's¹. 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 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 new lines 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 new lines.
- 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.
- If you put `pageheading` and `pagefooting` statements before the first `chapter` environment, Mint will create a couple of blank pages at the beginning of the document.
- The output driver for press files is not handling ligatures correctly on the last line of a paragraph. The character that follows a ligature is likely to be too close to it.
- `EDefs` are not being expanded at quite the right time, and they are interacting with the macrogenerator in all sorts of ill-mannered ways. Straightforward textual substitution is all right, though.
- Bullets occasionally become divorced from their text when an item occurs at the bottom of a page. (Later note: I think this is now fixed.)

¹ Scribe is the registered trade mark of Unilogic Ltd.

- Labels that are attached to `pageheading`, `pagefooting` and `foot` environments get lost.
 Attach them to the nearest environment that is intended for the main galley.
- The macrogenerator is a whole can of worms.

Part Two

Basic properties of documents

This part of the Reference Manual describes the lexical and syntactic properties of documents, and a few other basic properties of the system. The first section describes the input conventions — these are similar to those of Scribe, but there are sufficient differences that you should not expect Scribe documents to be processed by Mint without change. Mostly, however, the changes will be simple. The second section describes the syntax of documents. In general the syntax is sufficiently free that you do not need to be aware of it when you create a document. The next two sections describe the units of length that Mint uses, and the form of the error messages.

2.1 Input conventions

Mint allows several front-ends to coexist, and it will take lexemes from whichever 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 in the future. The input conventions for DP and Plot are described in section 13.1.1, and the input conventions for the mathematical environments are described in part 12; 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`), statement identifiers (such as `Include`), macro identifiers (such as `library`) and Mint commands (such as `begin`) are case-free, as are the formal attribute identifiers used to modify environments (such as `Need` and `width`), and the formal parameter identifiers of macros.
- 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, formal parameter identifiers of macros 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 (described in the next section), and not by attribute parameters of environments, 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. One environment where this may cause some surprises is the **quotation** environment: this does not allow blank lines within it. However, the **multiple** environment will do what Scribe users would expect **quotation** to do.
- If two environments have another environment between them, and there are no blank lines separating them, as for example in the case of

If two environments have another environment between them, and there are no blank lines separating them, 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 blank lines separate the environments, however, 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 blank lines separate the environments, however, the last paragraph will start with an indented line (provided that the document type has indentations for the first line).

- **Enter** and **Begin** are synonyms, as are **Leave** and **End**. Document environments do not need to have an **end**. The **make** statement performs two actions: first, it indicates which state file to use for the document; and second, it automatically creates a **begin** command for the document. See section 14.1.1 for more information.
- Mint will expand or contract spaces to fill out the lines of some environments (like the current one, for example). 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 new line, then the space is regarded as an end-of-sentence space, otherwise as a between-word space. For this reason 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 8.4.1 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 commands understand the block structure of a document, and at the end of the environment in which they occur the definitions and modifications are popped.

2.2 Document syntax

Mint has a more rigid notion of the syntax of a document than do many 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 nestings 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 2.2.6.

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.

2.2.1 Syntactic metalanguage

The syntax is described by a simplified version of a two-level grammar. A *notion* specifies a Mint environment, such as `centre` and `description`, and a *syntax class* specifies a collection of notions. For example, the syntax class `Headings` specifies the notions `MajorHeading`, `Heading` and `SubHeading`. A production rule is written in terms of syntax classes as

```
SyntaxClass = [SyntaxClass1 + SyntaxClass2 + ... + SyntaxClassn]
```

or

```
SyntaxClass = []
```

The first rule is an abbreviation for the production rules

```
SyntaxClass = SyntaxClassa*  
SyntaxClassa = SyntaxClass1 | SyntaxClass2 | ... | SyntaxClassn
```

which describes the environments that can nest within the environments described in the collection `SyntaxClass`; these are the environments specified by `SyntaxClass1`, `SyntaxClass2`, etc. The second rule specifies that the collection of environments specified by `SyntaxClass` have no other environments nested within them.

In addition to the production rules, the input language to Mint is described by the error-correcting rules. Currently these are somewhat crude (which is reflected by 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, ..., Notion_n}

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.

2.2.2 Pseudo-Syntactic properties

Several other properties are associated with the notions of a document; these are termed *pseudo-syntactic properties* since they affect the interpretation of a document but also have semantic properties. Two of them, **pack** and **blanklines**, are described in this section. **Pack** takes a value from (**true**, **false**, **skip**), 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. An environment that does not use **pack** sets its value to **skip**. **Blanklines** takes a value from (**ignored**, **kept**, **invalid**), and specifies whether consecutive blank lines in an environment are to be compressed into a single one or not. A value of **invalid** indicates that blank lines cannot occur in the environment. 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 3.1.4; **autoincrement**, which is described in section 2.2.2; and **autocrossref**, which is described in section 2.2.2².

2.2.3 Standard environments

The following are the standard environments. They are used in the same way as they are in Scribe. For example

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

The brief descriptions below indicate informally the properties of each environment. More details are given in section 4.2.

Abstract	This places a paragraph of text in the position occupied by an abstract on a title page.
Align	An environment like verbatim , which treats tabulations in a way that is useful for laying out tables. It is described more fully in section 4.5.3.

² Well, all right then. An environment can have a cross-reference string that gets automatically inserted into a galley, such as occurs when an annotation or a page command appears; such environments have an **autocrossref** that is not null. If **autoincrement** is true then it indicates that the dominating environment counter is to be incremented for each occurrence of the environment. The annotating environments have **autoincrements** that are true.

Annotation	An annotation is similar to a footnote, though at present it does not appear in the pages. This environment is used in the editorial annotation feature that will be described in a later edition of the manual.
Appendix	An appendix is similar to a chapter, except for the numbering style 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. (This environment can also be spelled center .)
Chapter	A chapter is preceded by a number, the style of which 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 creates a copyright notice, preceded by the copyright symbol, on the title page.
Default	The default environment type for all documents except slides . 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 4.5.4.
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	A ditem is what each of the paragraphs is in a description environment. You usually won't need to use this environment explicitly.
DP	This is used for DP drawings. Section 13.1.1 describes how to use this environment.
Enumerate	This produces a list of paragraphs, each preceded by a letter or number, depending on the depth of enumeration. (Paragraph is used here in the informal sense, not the sense defined below.)
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, or a mathematical formula, etc, or a diagram produced by Plot or LDP, 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 to flushleft , 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 specify explicitly an environment as being a gloss , otherwise it will be assumed to be a textpart .
Heading	This produces a centred heading in a bold typeface that is smaller than that used for majorheading .
Item	An item is what each of the paragraphs is in an itemize and enumerate environment. You usually won't need to use this environment explicitly.
Itemize	This produces a list of paragraphs, each preceded by a bullet.
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. There are two forms of manual: form 0 labels chapters with the word Chapter , form 1 labels them with Part . (This reference manual is using form 1.)
Maths	This environment uses special processing of its body to do high quality mathematical typesetting. Details are given in part 12. (The environment can also be spelled math .)
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 titlepage environment.
PageCommand	The pagecommand 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 8.4.1.
PageFooting	This environment, similar to the pageheading environment, places footings on the pages. It is described in more detail in section 8.4.1. (There should also be pageleftmargin and pagerightmargin . Watch this space.)

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 8.4.2.
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 13.1.1 describes how to use this environment.
Portion	This environment is used in the separate formatting facility. You would not normally use this directly, but instead you would use a statement. This facility isn't described yet.
PrefaceSection	A preface section will start a new page, and produce the heading in a large 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. (Maybe.)
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 on a title page.
Section	A section has a number, and is flushed left. It uses the same typeface as that used for <code>chapter</code> .
Slides	A document type without numbered sections, with spacings and fonts suitable for making overhead transparency slides. The default environment for slides is <code>verbatim</code> .
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 <code>chapter</code> and <code>section</code> .
Table	A table comprises a caption, followed by a paragraph of text, a mathematical formula, etc, or a diagram produced using Plot or DP.
Text	The default document type. It has no numbered sections, and occurs in two forms. Form 0 has wide-spaced lines, and form 1 has narrower lines.
TextPart	The <code>textpart</code> 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.

TitleBox	The environments that are allowed within the title box are majorheading , heading and centre .
TitlePage	This environment is used for laying out title pages. It controls the positions of the environments which nest within it.
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 that in which they are broken in the manuscript, unless the line is too long to fit within the margins. In this latter case it indents the broken line.

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

Note that there is no **equation** environment. The **maths** environment allows equations to be laid out.

2.2.4 Production rules common across several document types

In the description below, the syntax class is followed by the notions it produces, the default notions, the value of **Pack** and the value of **BlankLines**. There are several syntax classes that contain only one notion. To avoid the proliferation of identifiers, these syntax classes are shown as <Notion> when they occur on the right hand side of production rules.

The following syntax classes are used.

```

Headings = MajorHeading, Heading, SubHeading
Items = Commentary, Describe, Description, Enumerate, Figure, Itemize,
          Multiple, Portion, Table
Pages = PageCommand, PageHeading, PageFootting, PageOffset
Sections = Appendix, AppendixSection, Chapter3, Paragraph, PrefaceSection,
            Section, SubSection
Terminals = Align, Centre, Default, Display, DP, Example, FlushLeft,
              FlushRight, Format, Maths, ProgramExample, Plot, Quotation,
              TextPart, Verse, Verbatim
Titles = Abstract, CopyrightNotice, Notice, ResearchCredit, TitleBox

```

³ Chapter does not occur in all document types.

2.2.4.1 Document environment syntax

Article =	[Headings + Items + Sections ⁻ + Terminals + <TitlePage>]			
		{Default}	Skip	Invalid
Manual =	[Headings + Items + Sections ⁺ + Terminals + <TitlePage>]			
		{Default}	Skip	Invalid
Report =	[Headings + Items + Sections ⁺ + Terminals + <TitlePage>]			
		{Default}	Skip	Invalid
Slides =	[Headings + Items + Terminals]			
		{Verbatim}	Skip	Invalid
Text =	[Headings + Items + Terminals]			
		{Default}	Skip	Invalid
Thesis =	[Headings + Items + Sections ⁺ + Terminals + <TitlePage>]			
		{Default}	Skip	Invalid

Those document types with sections marked by + have the chapter environment, those with - do not.

2.2.4.2 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	Ignored
Plot =	[]	{}	True	Kept
ProgramExample =	[]	{}	False	Kept
Quotation =	[]	{}	True	Invalid
TextPart =	[]	{}	False	Kept
Verbatim =	[]	{}	False	Kept
Verse =	[]	{}	False	Kept

2.2.4.3 Heading environment syntax

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

2.2.4.4 Section environment syntax

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

2.2.4.5 Item environment syntax

Commentary =	[Terminals + Items + <Gloss>]	{TextPart}	Skip	Invalid
Describe =	[Terminals + Items]	{FlushLeft}	Skip	Invalid
Description =	[Terminals + Items + <DItem>]	{DItem}	Skip	Invalid
Enumerate =	[Terminals + Items + <Item>]	{Item}	Skip	Invalid
Figure =	[Terminals + Items + <Caption>]	{TextPart}	Skip	Invalid
Itemize =	[Terminals + Items + <Item>]	{Item}	Skip	Invalid
Multiple =	[Terminals + Items]	{Default}	Skip	Invalid
Table =	[Terminals + Items + <Caption>]	{Align}	Skip	Invalid
Caption =	[]	{}	True	Invalid
DItem =	[]	{}	True	Invalid
Gloss =	[]	{}	True	Invalid
Item =	[]	{}	True	Invalid

2.2.4.6 Title page environment syntax

Abstract =	[Terminal + Items]	{Default}	Skip	Invalid
CopyrightNotice =	[]	{}	True	Invalid
Notice =	[Terminal + Items]	{Default}	Skip	Invalid
ResearchCredit =	[Terminal + Items]	{Default}	Skip	Invalid
TitlePage =	[Titles]	{Notice}	Skip	Invalid
TitleBox =	[Terminals + Headings]	{Centre}	Skip	Invalid

2.2.4.7 Galley environment syntax

Annotation =	[Terminals + Items]	{Default}	Skip	Invalid
Foot =	[Terminals + Items]	{Default}	Skip	Invalid
Contents =	[Terminals + Items + Headings]	{Align}	Skip	Invalid
MAnnotation =	[<Annotation>]	{Annotation}	Skip	Invalid
MContents =	[<Contents>]	{Contents}	Skip	Invalid
MFoot =	[<Foot>]	{Foot}	Skip	Invalid
MRoot =	One of [<Article>], [<Manual>], [<Report>], [<Slides>], [<Text>], [<Thesis>]	{}	Skip	Invalid

2.2.4.8 Page environment syntax

PageCommand =	[]	{}	True	Invalid
PageHeading =	[Terminals]	{Centre}	Skip	Invalid
PageFootning =	[Terminals]	{Centre}	Skip	Invalid
PageOffset =	[]	{Centre}	True	Invalid
MOddsandSods =	[Pages]	{}	Skip	Invalid

2.2.5 Document Syntax

Each document type has a different syntax. The syntax for a document consists of the sum of the syntaxes needed for each of the galleys into which the document will be formatted. More information about galleys is given in part 3.

Article Syntax	[<Article> + Pages + <Foot> + <Annotation>]
Manual Syntax	[<Manual> + Pages + <Foot> + <Annotation> + <Contents>]
Report Syntax	[<Report> + Pages + <Foot> + <Annotation> + <Contents>]
Slides Syntax	[<Slides> + Pages]
Text Syntax	[<Text> + Pages + <Foot> + <Annotation>]
Thesis Syntax	[<Thesis> + Pages + <Foot> + <Annotation> + <Contents>]

2.2.6 Altering the syntax

Sometimes you will discover that the Mint syntax is too restrictive to allow some layouts that you want. For example, you might want an **itemize** environment in a titlebox. Mint provides a means of altering the syntax from within the .Mss file, using the statements: **AddClass**, which adds a new right hand side to a production rule; **RemClass**, 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 **AddClass** 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

```
@AddClass (LHS, RHS)  
@RemClass (LHS, RHS)
```

which add and remove environments from the syntax class LHS, and

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

which add and remove defaults from the environment LHS.

The new syntax allows a greater variety of document structures than was previously available. Only in extreme cases should it be necessary to alter the syntax. For example, the syntax in the previous version of Mint did not allow you to incorporate commentaries into figures. It now does, so that:

```

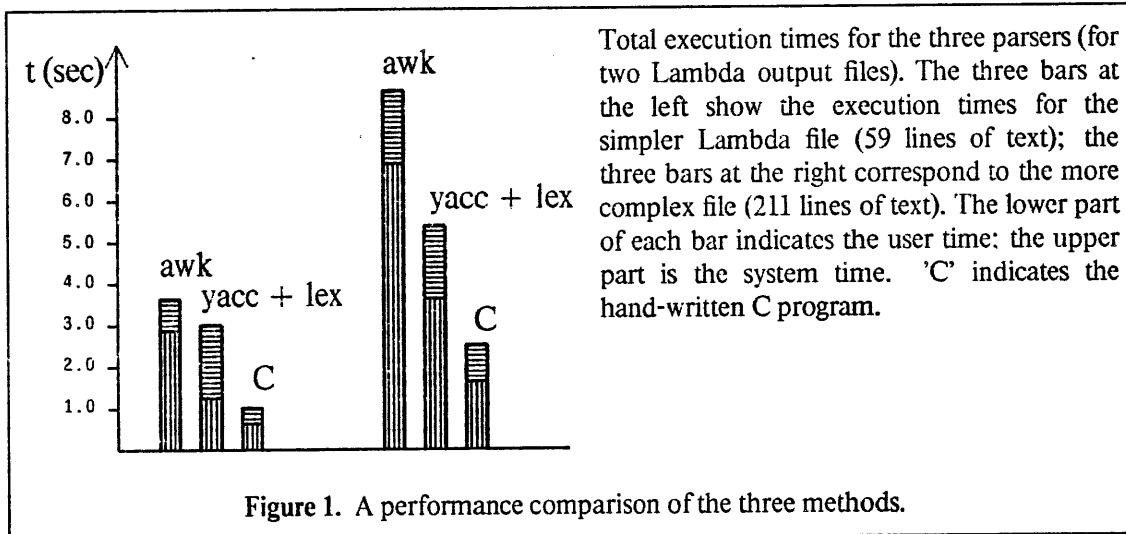
@begin{figure, width = 6in, borderstyle width1, border = 0.1 in}
@begin[commentary]
@begin[dp, width = 3 in]@include[bargraph.dp]@end[dp]

@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}

```

produces the following:



2.3 Units of length

Mint has a fairly precise notion of units of length, and the units must be specified 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 *internal units*; and *layout relative units*, measured in terms of the sizes of pages, or characters in some font.

Mint classifies device relative and layout relative units into horizontal and vertical measures. Wherever it says, in this manual, something like "Mint expects a vertical unit", then Mint will require the unit of

length to be an appropriate vertical unit, such as a `line`, or a unit that can be used in the vertical direction, such as an `inch`. In addition, during galley creation it is not appropriate to talk about the size of a page <Oh dear, that seems wrong>, 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 in these circumstances.

The units are as follows.

2.3.1 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. $1 \text{ point} = 0.013837 \text{ inches}$.
<code>pica</code>	Also <code>picas</code> . Ditto. $1 \text{ pica} = 0.166044 \text{ inch}$.
<code>cm</code>	Also <code>cms</code> . $1 \text{ cm} = 0.3937 \text{ inch}$.
<code>mica</code>	Also <code>micas</code> . $1 \text{ mica} = 0.001 \text{ cm} = 0.0003937 \text{ inch}$.

2.3.2 Internal units

Mint assumes that there is some device-dependent unit of distance, the `iu` (internal unit), which may not necessarily be the same in absolute units in the *X* and *Y* directions. Mint uses these units internally, and converts all other units into them. The absolute size of an internal unit is specified when a device class is declared. Normally you will never use these units, though they are generated by the `charinfo` statement.

2.3.3 Relative lengths

There are two sets of relative lengths. During galley preparation, the unit of distance is the `em`, a traditional typesetting measure of distance. An em describes the size of the letter M, which in many classical fonts has the same width as height. Since many modern fonts do not have this property, Mint uses two values for the em, according to whether it is used vertically or horizontally: this has some useful features when describing font properties. In the horizontal direction the size of an em is the value of the `EmWidth` parameter for the font, if it has been defined, otherwise it is the value of $XX - X0$ for the character 'M'. In the vertical direction the size of the em is the value of the `EmHeight` parameter for the font, if it has been defined, otherwise it is the value of YY for the character 'M'. See section 5.4 for more details. When used

as a unit of measure in a box, an **em** refers to the default font for the box. Units in this measure can be expressed as **em** or **ems**, or as **quad** or **quads**⁴.

The unit of distance **line** and **lines** measures the distance between the baselines of consecutive lines in a box — this distance is the size of a vertical **em** plus the gap that separates the slugs.

During page layout, the relevant measures are **pagewidth**, the width of the page for the target device, and **pageheight** respectively.

2.3.4 Scaling

Documents are frequently reproduced by photographic processes which allow magnification or (more frequently) reduction. It is useful to be able to describe lengths in terms of the photographically reproduced document, rather than in terms of the original from the printer. This means that if you want to have a diagram that is to be 3 inches wide after reproduction at 90% reduction, you can describe it as 3 inches wide, even though it is actually 3.33 inches wide on the output page.

This scaling should be done not only to the units of length, but also to the representation of the fonts that are used in the document. Unfortunately scaling of fonts cannot be done easily for raster devices like the Perq and Dover; fortunately Metafont fonts exist which, although nominally of one point size, are actually magnified. To produce a scaled document you must therefore set the font families with the appropriate scaled fonts, and use the **setscale** statement to set the scaling for units of length. For example

```
@setscale(1.11)
```

will scale all the units appropriately for 90% reduction.

2.3.5 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

⁴ Note that a **quad** does not measure the point size of the font. This will be fixed in future versions of Mint, and I will probably rename the **em** to avoid confusion with its traditional meaning.

below parameter, and increase the **need** parameter by three centimeters. The **need** parameter can be set to **all**, when it specifies the vertical size of the box.

2.4 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 by responding appropriately to the prompt, but the effects are unpredictable. You should report Heresies and Fatal Errors to me using the report mechanism described in the User Manual.

Error messages appear in the typescript 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.

Part Three

Galleys

Galleys are the principal data structures within Mint. They specify the formatting rules that will be applied within 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, galleys are also formatting processes. The interactions between these processes 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 section 3.1 I describe the principal properties of galleys; section 3.2 describes the default properties of the galleys that are set up automatically when Mint starts.

3.1 Galley components

The properties of a galley are specified by several collections of information. The information that specifies how the galley is to format the document is collected into *procedure families*, which can be declared and manipulated within Mint. Since a procedure family specifies the behaviour of a galley when interpreting the manuscript, associating the same procedure family with several galleys is sometimes useful.

The information that specifies which fonts will be used in the galley is collected together into a *font family*, which is also an object that can be declared and manipulated within Mint, and which can similarly be associated with several galleys.

Global values that describe the *style* of a galley are collected into a styles object. This is a collection of values, describing for example, several values for margins, the `above` and `below`, the gap between lines, etc., that can get extracted by each environment. Styles objects can also be associated simultaneously with several galleys.

To specify a galley fully several other items of information are required. These are described together in section 3.1.5.

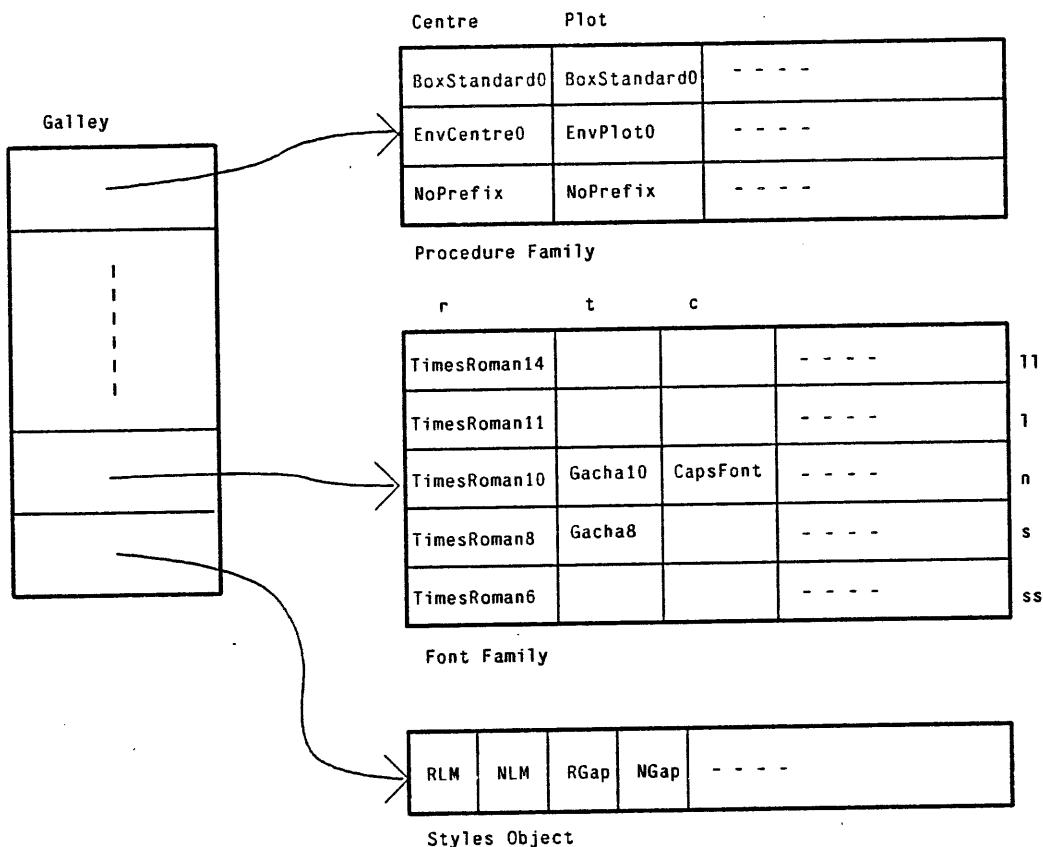


Figure 2. Structure of a galley

3.1.1 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 3.1.4 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 three parts, a *procedure indicator* an *environment indicator* and a *prefix 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, and the prefix indicator specifies what prefix (or

postfix) string will be generated automatically for the environment. There are fewer procedures than environment parameter sets, because several environments can be analysed using the same procedure. For example, the `FlushLeft`, the `Verbatim` and the `Gloss` environments are analysed by the same procedure, because the differences between these environments are captured completely by the differences between the environment parameters. See section 4.3 for a description of the procedures, see section 4.1 for a description of the box environment parameters, and see section 7.5 for a description of the prefixes.

A new procedure family is created by the statement

```
@NewProcFamily (FloatingFigures)
```

which creates a new vector, named `FloatingFigures`, and initializes all the entries to null. (You should also be able to copy procedure families, but you can't yet.)

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

```
@AssocProc (FloatingFigures, DP, BoxStandard0, EnvDPO, NoPrefix)
```

specifies that the `FloatingFigures` procedure family is able to handle the `DP` environment, using the procedure `BoxStandard0` and the environment parameters `EnvDPO`, and that no prefix string is to be used. 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 Mint to do to help you out. You are probably in good shape adding terminal environments, though.

3.1.2 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 5.1, which explains how to add fonts to the font family, and how to manufacture fonts from existing ones.

3.1.3 Styles

A *styles object* is a collection of values that is associated with a galley when it is created. When an environment starts, the environment parameters can be selected from the styles object (possibly after transformations), though an environment can obtain its values in other ways as well. The values in the styles object, together with the inheritance rules used by the environments, determine to a large extent the appearance of documents. For example, a styles object contains two values, the `ngap` and `rgap`, that are used by most environments to specify the leading between lines. That is, most environments specify their `gap` attribute to be either `ngap` or `rgap`; by using appropriate values in the styles object of a galley you can control the inter-line spacing throughout the document. In fact the major difference between `text`, `form 0` and `text`, `form 1` is in these two style values.

A styles object is created by the `makeStyle` statement; this takes an identifier, a device identifier and a collection of values. Section 14.3.3.1 shows an example of the statement, and section 3.2.3 lists the values of the style objects used in the principal document types.

3.1.4 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 of the document. The following environments are specified as dominating in the current system.

<code>Foot</code> <code>PageOffset</code>	<code>Annotation</code> <code>PageCommand</code>	<code>PageHeading</code>	<code>PageFootting</code>
--	---	--------------------------	---------------------------

3.1.5 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, you have to specify the procedure family,

the font family and the style values that are associated with it, and the environment whose procedure is at the bottom of the invocation stack. An additional parameter specifies whether the galley becomes active immediately or not (the only galley which does not is the **Contents** galley, which is started automatically after the other galleys).

A galley is created by the statement **NewGalley**, as in the statement

```
@NewGalley (FootNotes, Annotations, Fonts1, FootPF, Style0, MFoot, True)
```

This statement creates a new galley called **FootNotes**, which belongs to the class of galleys called **Annotations**⁵, with font family **Fonts1**, procedure family **FootPF**, styles **Style0**, and with the procedure **MFoot** at the bottom of the invocation stack. The procedure will be invoked immediately. Mint will check that the device specified in the styles object belongs to the class of devices specified in the font family; the galley is made for that particular device. Usually the galley will suspend immediately, because it requires lexemes to proceed, and it 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 again 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.

3.1.6 Changing galley properties

The properties of a galley can be changed after it has been created. It is possible to use **assocfont** and **assocproc** to change individual entries in the font and procedure families associated with a galley. Since the same family may be associated with several galleys, each galley will change its properties. It is not possible to change individual entries in the styles object associated with a galley, except by using the **make** statement. See section 4.1.2.1 for an explanation of how to do this.

Alternatively, the whole of the font family, procedure family or styles object can be changed. The following statements perform these actions.

```
@SetFontF (Main, KanjiFonts)
@SetProcF (Main, KanjiProcs)
@SetStyle (Main, Scroll)
```

⁵ It is no place here to describe the different classes of galley. Mint handles each class differently. The classes are **main**, **annotations**, **miscellaneous** and **contents**.

These statements will change the font family, procedure family and styles object to those specified (they should have been created already, of course). The font family and the procedure family can be changed at any time; however, it does not make sense to change the styles object after there are slugs in the galley, so normally you will want to put the `set style` statement at the beginning of the document, or in the `.Defs` file (section 14.1). Note that you can change the device associated with the galley, but it must belong to the same device class as the original device (see section 11.1).

3.2 Standard Galley Properties

This section describes the properties of the standard galleys that are made available by using the standard state files (section 14.1.1) in the current version of Mint. I describe several procedure families, font families, prefixes and styles before showing in section 3.2.4 how each of the galleys is composed.

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.

3.2.1 Procedure families

The following procedure families are defined. The name of each environment is followed by the procedure indicator and environment indicator that the environment uses. See section 3.1.1 for the meanings of these terms.

3.2.1.1 Procedure family MainPFO

<code>Centre</code>	<code>BoxStandard0</code>	<code>EnvCentre0</code>	<code>NoPrefix</code>
<code>Display</code>	<code>BoxStandard0</code>	<code>EnvDisplay0</code>	<code>NoPrefix</code>
<code>Example</code>	<code>BoxStandard0</code>	<code>EnvExample0</code>	<code>NoPrefix</code>
<code>ProgramExample</code>	<code>BoxStandard0</code>	<code>EnvProgramEx0</code>	<code>NoPrefix</code>
<code>FlushLeft</code>	<code>BoxStandard0</code>	<code>EnvFlushLeft0</code>	<code>NoPrefix</code>
<code>FlushRight</code>	<code>BoxStandard0</code>	<code>EnvFlushRight0</code>	<code>NoPrefix</code>
<code>Verbatim</code>	<code>BoxStandard0</code>	<code>EnvVerbatim0</code>	<code>NoPrefix</code>
<code>Format</code>	<code>BoxStandard0</code>	<code>EnvFormat0</code>	<code>NoPrefix</code>
<code>Quotation</code>	<code>BoxStandard0</code>	<code>EnvQuotation0</code>	<code>NoPrefix</code>
<code>Default</code>	<code>BoxStandard0</code>	<code>EnvDefault0</code>	<code>NoPrefix</code>
<code>Align</code>	<code>BoxStandard0</code>	<code>EnvAlign0</code>	<code>NoPrefix</code>
<code>Plot</code>	<code>BoxStandard0</code>	<code>EnvPlot0</code>	<code>NoPrefix</code>
<code>DP</code>	<code>BoxStandard0</code>	<code>EnvDPO</code>	<code>NoPrefix</code>
<code>Verse</code>	<code>BoxStandard0</code>	<code>EnvVerse0</code>	<code>NoPrefix</code>
<code>Maths</code>	<code>BoxMaths0</code>	<code>EnvMaths0</code>	<code>PostfixEqn</code>

MajorHeading	BoxStandard0	EnvMajorHeading0	NoPrefix
Heading	BoxStandard0	EnvHeading0	NoPrefix
SubHeading	BoxStandard0	EnvSubHeading0	NoPrefix
PrefaceSection	BoxStandard0	EnvPrefaceSec0	NoPrefix
Chapter	BoxSectionEnv0	EnvChapter0	PrefixChapter
Section	BoxSectionEnv0	EnvSection0	PrefixSection
SubSection	BoxSectionEnv0	EnvSubSection0	PrefixSubSection
Paragraph	BoxSectionEnv0	EnvParagraph0	PrefixParagraph
Appendix	BoxSectionEnv0	EnvAppendix0	PrefixAppendix
AppendixSection	BoxSectionEnv0	EnvAppendixSec0	PrefixAppendixSec
Itemize	BoxItemize0	EnvItemize0	NoPrefix
Item	BoxStandard0	EnvItem0	NoPrefix
Description	BoxMultiple0	EnvDescription0	NoPrefix
DItem	BoxStandard0	EnvDItem0	NoPrefix
Commentary	BoxCommentary0	EnvCommentary0	NoPrefix
TextPart	BoxStandard0	EnvTextPart0	NoPrefix
Gloss	BoxStandard0	EnvGloss0	NoPrefix
Figure	BoxFigure0	EnvFigure0	PrefixFigure
Table	BoxTable0	EnvTable0	PrefixTable
Caption	BoxCaption0	EnvCaption0	NoPrefix
Enumerate	BoxEnumerate0	EnvEnumerate0	NoPrefix
Describe	BoxDescribe0	EnvDescribe0	NoPrefix
Multiple	BoxMultiple0	EnvMultiple0	NoPrefix
TitlePage	BoxMultiple0	EnvTitlePage0	NoPrefix
TitleBox	BoxMultiple0	EnvTitleBox0	NoPrefix
ResearchCredit	BoxMultiple0	EnvResearchCr0	NoPrefix
Abstract	BoxMultiple0	EnvAbstract0	NoPrefix
Notice	BoxMultiple0	EnvNotice0	NoPrefix
Portion	BoxPortion0	EnvPortion0	NoPrefix
CopyrightNotice	BoxSectionEnv0	EnvCopyrightN0	PrefixCopyrtN
Dokument	BoxGalley0	EnvDokument0	NoPrefix
MRoot	BoxRoot0	EnvRoot0	NoPrefix

3.2.1.2 Procedure family MainPF1

Centre	BoxStandard0	EnvCentre0	NoPrefix
Display	BoxStandard0	EnvDisplay0	NoPrefix
Example	BoxStandard0	EnvExample0	NoPrefix
ProgramExample	BoxStandard0	EnvProgramEx0	NoPrefix
FlushLeft	BoxStandard0	EnvFlushLeft0	NoPrefix
FlushRight	BoxStandard0	EnvFlushRight0	NoPrefix
Verbatim	BoxStandard0	EnvVerbatim0	NoPrefix
Format	BoxStandard0	EnvFormat0	NoPrefix
Quotation	BoxStandard0	EnvQuotation0	NoPrefix
Default	BoxStandard0	EnvDefault0	NoPrefix
Align	BoxStandard0	EnvAlign0	NoPrefix
Plot	BoxStandard0	EnvPlot0	NoPrefix
DP	BoxStandard0	EnvDP0	NoPrefix
Verse	BoxStandard0	EnvVerse0	NoPrefix
Maths	BoxMaths0	EnvMaths0	PostfixEqn
MajorHeading	BoxStandard0	EnvMajorHeading0	NoPrefix
Heading	BoxStandard0	EnvHeading0	NoPrefix
SubHeading	BoxStandard0	EnvSubHeading0	NoPrefix
PrefaceSection	BoxStandard0	EnvPrefaceSec0	NoPrefix
Section	BoxSectionEnv0	EnvSection0	PrefixSection
SubSection	BoxSectionEnv0	EnvSubSection0	PrefixSubSection

Paragraph	BoxSectionEnv0	EnvParagraph0	PrefixParagraph
Appendix	BoxSectionEnv0	EnvAppendix1	PrefixAppendix
AppendixSection	BoxSectionEnv0	EnvAppendixSec1	PrefixAppendixSec
Itemize	BoxItemize0	EnvItemize0	NoPrefix
Item	BoxStandard0	EnvItem0	NoPrefix
Description	BoxMultiple0	EnvDescription0	NoPrefix
DItem	BoxStandard0	EnvDItem0	NoPrefix
Commentary	BoxCommentary0	EnvCommentary0	NoPrefix
TextPart	BoxStandard0	EnvTextPart0	NoPrefix
Gloss	BoxStandard0	EnvGloss0	NoPrefix
Figure	BoxFigure0	EnvFigure0	PrefixFigure
Table	BoxTable0	EnvTable0	PrefixTable
Caption	BoxCaption0	EnvCaption0	NoPrefix
Enumerate	BoxEnumerate0	EnvEnumerate0	NoPrefix
Describe	BoxDescribe0	EnvDescribe0	NoPrefix
Multiple	BoxMultiple0	EnvMultiple0	NoPrefix
TitlePage	BoxMultiple0	EnvTitlePage0	NoPrefix
TitleBox	BoxMultiple0	EnvTitleBox0	NoPrefix
ResearchCredit	BoxMultiple0	EnvResearchCr0	NoPrefix
Abstract	BoxMultiple0	EnvAbstract0	NoPrefix
Notice	BoxMultiple0	EnvNotice0	NoPrefix
Portion	BoxPortion0	EnvPortion0	NoPrefix
CopyrightNotice	BoxSectionEnv0	EnvCopyrightNO	PrefixCopyrtN
Dokument	BoxGalley0	EnvDokument0	NoPrefix
MRoot	BoxRoot0	EnvRoot0	NoPrefix

3.2.1.3 Procedure family MainPF2

Centre	BoxStandard0	EnvCentre0	NoPrefix
Display	BoxStandard0	EnvDisplay0	NoPrefix
Example	BoxStandard0	EnvExample0	NoPrefix
ProgramExample	BoxStandard0	EnvProgramEx0	NoPrefix
FlushLeft	BoxStandard0	EnvFlushLeft0	NoPrefix
FlushRight	BoxStandard0	EnvFlushRight0	NoPrefix
Verbatim	BoxStandard0	EnvVerbatim0	NoPrefix
Format	BoxStandard0	EnvFormat0	NoPrefix
Quotation	BoxStandard0	EnvQuotation0	NoPrefix
Default	BoxStandard0	EnvDefault0	NoPrefix
Align	BoxStandard0	EnvAlign0	NoPrefix
Plot	BoxStandard0	EnvPlot0	NoPrefix
DP	BoxStandard0	EnvDP0	NoPrefix
Verse	BoxStandard0	EnvVerse0	NoPrefix
Maths	BoxMaths0	EnvMaths0	PostfixEqn
MajorHeading	BoxStandard0	EnvMajorHeading0	NoPrefix
Heading	BoxStandard0	EnvHeading0	NoPrefix
SubHeading	BoxStandard0	EnvSubHeading0	NoPrefix
Itemize	BoxItemize0	EnvItemize0	NoPrefix
Item	BoxStandard0	EnvItem0	NoPrefix
Description	BoxMultiple0	EnvDescription0	NoPrefix
DItem	BoxStandard0	EnvDItem0	NoPrefix
Commentary	BoxCommentary0	EnvCommentary0	NoPrefix
TextPart	BoxStandard0	EnvTextPart0	NoPrefix
Gloss	BoxStandard0	EnvGloss0	NoPrefix
Figure	BoxFigure0	EnvFigure0	PrefixFigure
Table	BoxTable0	EnvTable0	PrefixTable
Caption	BoxCaption0	EnvCaption0	NoPrefix

Enumerate	BoxEnumerate0	EnvEnumerate0	NoPrefix
Describe	BoxDescribe0	EnvDescribe0	NoPrefix
Multiple	BoxMultiple0	EnvMultiple0	NoPrefix
Portion	BoxPortion0	EnvPortion0	NoPrefix
Dokument	BoxGalley0	EnvDokument0	NoPrefix
MRoot	BoxRoot0	EnvRoot0	NoPrefix

3.2.1.4 Procedure family FootPF0

Centre	BoxStandard0	EnvCentre0	NoPrefix
Display	BoxStandard0	EnvDisplay0	NoPrefix
Example	BoxStandard0	EnvExample0	NoPrefix
ProgramExample	BoxStandard0	EnvProgramEx0	NoPrefix
FlushLeft	BoxStandard0	EnvFlushLeft0	NoPrefix
FlushRight	BoxStandard0	EnvFlushRight0	NoPrefix
Verbatim	BoxStandard0	EnvVerbatim0	NoPrefix
Format	BoxStandard0	EnvFormat0	NoPrefix
Quotation	BoxStandard0	EnvQuotation0	NoPrefix
Default	BoxStandard0	EnvDefault0	NoPrefix
Align	BoxStandard0	EnvAlign0	NoPrefix
Plot	BoxStandard0	EnvPlot0	NoPrefix
DP	BoxStandard0	EnvDPO	NoPrefix
Verse	BoxStandard0	EnvVerse0	NoPrefix
Maths	BoxMaths0	EnvMaths0	PostfixEqn
Itemize	BoxItemize0	EnvItemize0	NoPrefix
Item	BoxStandard0	EnvItem0	NoPrefix
Description	BoxMultiple0	EnvDescription0	NoPrefix
DItem	BoxStandard0	EnvDItem0	NoPrefix
Commentary	BoxCommentary0	EnvCommentary0	NoPrefix
TextPart	BoxStandard0	EnvTextPart0	NoPrefix
Gloss	BoxStandard0	EnvGloss0	NoPrefix
Figure	BoxFigure0	EnvFigure0	PrefixFigure
Table	BoxTable0	EnvTable0	PrefixTable
Caption	BoxCaption0	EnvCaption0	NoPrefix
Enumerate	BoxEnumerate0	EnvEnumerate0	NoPrefix
Describe	BoxDescribe0	EnvDescribe0	NoPrefix
Multiple	BoxMultiple0	EnvMultiple0	NoPrefix
Foot	BoxCrossRef0	EnvFoot0	NoPrefix
MFoot	BoxGalley0	EnvMFoot0	NoPrefix

3.2.1.5 Procedure family ContPF0

Centre	BoxStandard0	EnvCentre0	NoPrefix
Display	BoxStandard0	EnvDisplay0	NoPrefix
Example	BoxStandard0	EnvExample0	NoPrefix
ProgramExample	BoxStandard0	EnvProgramEx0	NoPrefix
FlushLeft	BoxStandard0	EnvFlushLeft0	NoPrefix
FlushRight	BoxStandard0	EnvFlushRight0	NoPrefix
Verbatim	BoxStandard0	EnvVerbatim0	NoPrefix
Format	BoxStandard0	EnvFormat0	NoPrefix
Quotation	BoxStandard0	EnvQuotation0	NoPrefix
Default	BoxStandard0	EnvDefault0	NoPrefix
Align	BoxStandard0	EnvAlign0	NoPrefix
Plot	BoxStandard0	EnvPlot0	NoPrefix

DP	BoxStandard0	EnvDPO	NoPrefix
Verse	BoxStandard0	EnvVerse0	NoPrefix
Maths	BoxMaths0	EnvMaths0	NoPrefix
MajorHeading	BoxStandard0	EnvMajorHeading0	NoPrefix
Heading	BoxStandard0	EnvHeading0	NoPrefix
SubHeading	BoxStandard0	EnvSubHeading0	NoPrefix
Itemize	BoxItemize0	EnvItemize0	NoPrefix
Item	BoxStandard0	EnvItem0	NoPrefix
Description	BoxMultiple0	EnvDescription0	NoPrefix
DItem	BoxStandard0	EnvDItem0	NoPrefix
Commentary	BoxCommentary0	EnvCommentary0	NoPrefix
TextPart	BoxStandard0	EnvTextPart0	NoPrefix
Gloss	BoxStandard0	EnvGloss0	NoPrefix
Figure	BoxFigure0	EnvFigure0	NoPrefix
Table	BoxTable0	EnvTable0	NoPrefix
Caption	BoxCaption0	EnvCaption0	NoPrefix
Enumerate	BoxEnumerate0	EnvEnumerate0	NoPrefix
Describe	BoxDescribe0	EnvDescribe0	NoPrefix
Multiple	BoxMultiple0	EnvMultiple0	NoPrefix
Contents	BoxMultiple0	EnvContents0	NoPrefix
MContents	BoxGalley0	EnvMContents0	NoPrefix

3.2.1.6 Procedure family OddsPFO

Centre	BoxStandard0	EnvCentre0	NoPrefix
Display	BoxStandard0	EnvDisplay0	NoPrefix
Example	BoxStandard0	EnvExample0	NoPrefix
ProgramExample	BoxStandard0	EnvProgramEx0	NoPrefix
FlushLeft	BoxStandard0	EnvFlushLeft0	NoPrefix
FlushRight	BoxStandard0	EnvFlushRight0	NoPrefix
Verbatim	BoxStandard0	EnvVerbatim0	NoPrefix
Format	BoxStandard0	EnvFormat0	NoPrefix
Quotation	BoxStandard0	EnvQuotation0	NoPrefix
Default	BoxStandard0	EnvDefault0	NoPrefix
Align	BoxStandard0	EnvAlign0	NoPrefix
Plot	BoxStandard0	EnvPlot0	NoPrefix
DP	BoxStandard0	EnvDPO	NoPrefix
Verse	BoxStandard0	EnvVerse0	NoPrefix
Maths	BoxMaths0	EnvMaths0	PostfixEqn
PageHeading	BoxCrossRef0	EnvPageHeading0	NoPrefix
PageFootning	BoxCrossRef0	EnvPageFootning0	NoPrefix
PageOffset	BoxCrossRef0	EnvPageOffset0	NoPrefix
PageCommand	BoxCRTerm0	EnvPageCommand0	NoPrefix
MOddsandSods	BoxGalley0	EnvMOddsandSods0	NoPrefix

3.2.2 Font families

The following font families are defined.

Table 1. PerqScreen font family fonts0 (Main and contents galleys)

Face code	Font size				
	ss	s	n	1	11
r	-	Gacha9	TimesRoman12	-	-
i	-	-	TimesRoman12i	-	-
b	-	Gacha9	TimesRoman12b	TimesRoman14	TimesRoman18
c	-	-	-	-	-
g	-	-	-	-	-
t	-	Gacha9	Gacha12	-	-
p	-	-	-	-	-
z	-	-	-	-	-

Table 2. PressFile font family fonts0 (Main and contents galleys)

Face code	Font size				
	ss	s	n	1	11
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. PerqScreen font family fonts1 (Annotations and footnotes galleys)

Face code	Font size				
	ss	s	n	1	11
r	-	-	Gacha9	-	-
i	-	-	-	-	-
b	-	-	-	-	-
c	-	-	-	-	-
g	-	-	-	-	-
t	-	-	-	-	-
p	-	-	-	-	-
z	-	-	-	-	-

Table 4. PressFile font family fonts1 (Annotations and footnotes galley)

Face code	ss	s	Font size n	1	11
r	-	TimesRoman6	TimesRoman8	-	-
i	-	TimesRoman6i	TimesRoman8i	-	-
b	-	TimesRoman6b	TimesRoman8b	-	-
c	-	-	-	-	-
g	-	-	-	-	-
t	-	-	-	-	-
p	-	-	-	-	-
z	-	-	-	-	-

Table 5. PerqScreen font family fonts2 (Oddsandsods galley)

Face code	ss	s	Font size n	1	11
r	-	Gacha9	TimesRoman12	-	-
i	-	-	TimesRoman12i	-	-
b	-	-	TimesRoman12b	-	-
c	-	-	-	-	-
g	-	-	-	-	-
t	-	-	Gacha12	-	-
p	-	-	-	-	-
z	-	-	-	-	-

Table 6. PressFile font family fonts2 (Oddsandsods galley)

Face code	ss	s	Font size n	1	11
r	-	TimesRoman8	TimesRoman10	-	-
i	-	-	TimesRoman10i	-	-
b	-	-	TimesRoman10b	-	-
c	-	-	-	-	-
g	-	-	-	-	-
t	-	-	Gacha10	-	-
p	-	-	-	-	-
z	-	-	-	-	-

Table 7. PerqScreen font family fonts3 (Slides main galley)

Face code	ss	s	Font size n	1	11
r	TimesRoman12	TimesRoman12	TimesRoman14	TimesRoman18	TimesRoman18
i	-	-	-	-	-
b	-	TimesRoman12b	TimesRoman14	TimesRoman18	TimesRoman18
c	-	-	-	-	-
g	-	-	-	-	-
t	-	-	-	-	-
p	-	-	-	-	-
z	-	-	-	-	-

Table 8. PressFile font family fonts3 (Slides main galley)

Face code	Font size				
	ss	s	n	l	ll
r	Helvetica12	Helvetica14	Helvetica18	-	-
i	-	Helvetica14i	Helvetica18i	-	-
b	-	Helvetica14b	Helvetica18b	Helvetica24	Helvetica30
c	-	-	-	-	-
g	-	-	-	-	-
t	-	-	Helvetica18	-	-
p	-	-	-	-	-
z	-	-	-	-	-

Table 9. PerqScreen font family fonts4 (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. PressFile font family fonts4 (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	-	-	-	-	-

3.2.3 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 4.1.2.1 for a description of their actions. The following two styles objects 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 styles objects 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

3.2.4 The galley parameters for the document types

3.2.4.1 Text, form 0

The following galleys are defined.

```
@NewGalley (Main, Main, Fonts0, MainPFx, Style1, MRoot, True)
@NewGalley (FootNotes, Annotations, Fonts1, FootPFO, Style0, MFoot, True)
@NewGalley (Annotations, Annotations, Fonts1, NotePFO, Style0, MAnnotation,
            True)
@NewGalley (OddsandSods, Miscellaneous, Fonts2, OddsPFO, Style1,
            MOddsandSods, True)
```

3.2.4.2 Text, form 1

The following galleys are defined.

```
@NewGalley (Main, Main, Fonts0, MainPFx, Style0, MRoot, True)
@NewGalley (FootNotes, Annotations, Fonts1, FootPFO, Style0, MFoot, True)
@NewGalley (Annotations, Annotations, Fonts1, NotePFO, Style0, MAnnotation,
            True)
@NewGalley (OddsandSods, Miscellaneous, Fonts2, OddsPFO, Style0,
            MOddsandSods, True)
```

3.2.4.3 Report, form 0

The following galleys are defined.

```
@NewGalley (Main, Main, Fonts0, MainPFx, Style0, MRoot, True)
@NewGalley (FootNotes, Annotations, Fonts1, FootPF0, Style0, MFoot, True)
@NewGalley (Annotations, Annotations, Fonts1, NotePF0, Style0, MAnnotation,
    True)
@NewGalley (OddsandSods, Miscellaneous, Fonts2, OddsPF0, Style0,
    MOddsandSods, True)
@NewGalley (Contents, Contents, Fonts0, ContPF0, Style0, MContents, False)
```

3.2.4.4 Article, form 0

The following galleys are defined.

```
@NewGalley (Main, Main, Fonts0, MainPFx, Style0, MRoot, True)
@NewGalley (FootNotes, Annotations, Fonts1, FootPF0, Style0, MFoot, True)
@NewGalley (Annotations, Annotations, Fonts1, NotePF0, Style0, MAnnotation,
    True)
@NewGalley (OddsandSods, Miscellaneous, Fonts2, OddsPF0, Style0,
    MOddsandSods, True)
```

3.2.4.5 Thesis, form 0

The following galleys are defined.

```
@NewGalley (Main, Main, Fonts0, MainPFx, Style0, MRoot, True)
@NewGalley (FootNotes, Annotations, Fonts1, FootPF0, Style0, MFoot, True)
@NewGalley (Annotations, Annotations, Fonts1, NotePF0, Style0, MAnnotation,
    True)
@NewGalley (OddsandSods, Miscellaneous, Fonts2, OddsPF0, Style0,
    MOddsandSods, True)
@NewGalley (Contents, Contents, Fonts0, ContPF0, Style0, MContents, False)
```

3.2.4.6 Slides, form 0

The following galleys are defined.

```
@NewGalley (Main, Main, Fonts3, MainPFx, Style1, MRoot, True)
@NewGalley (OddsandSods, Miscellaneous, Fonts4, OddsPF0, Style1,
    MOddsandSods, True)
```

3.2.4.7 Manual, form 0

The following galleys are defined.

```
@NewGalley (Main, Main, Fonts0, MainPFx, Style0, MRoot, True)
@NewGalley (FootNotes, Annotations, Fonts1, FootPF0, Style0, MFoot, True)
@NewGalley (Annotations, Annotations, Fonts1, NotePF0, Style0, MAnnotation,
             True)
@NewGalley (OddsandSods, Miscellaneous, Fonts2, OddsPF0, Style0,
             MOddsandSods, True)
@NewGalley (Contents, Contents, Fonts0, ContPF0, Style0, MContents, False)
```

3.2.4.8 Manual, form 1

The following galleys are defined.

```
@NewGalley (Main, Main, Fonts0, MainPFx, Style0, MRoot, True)
@NewGalley (FootNotes, Annotations, Fonts1, FootPF0, Style0, MFoot, True)
@NewGalley (Annotations, Annotations, Fonts1, NotePF0, Style0, MAnnotation,
             True)
@NewGalley (OddsandSods, Miscellaneous, Fonts2, OddsPF0, Style0,
             MOddsandSods, True)
@NewGalley (Contents, Contents, Fonts0, ContPF0, Style0, MContents, False)
```

Part Four

Boxes and Slugs

Boxes are the subcomponents of galleys. They correspond to paragraphs, figures, tables, etc, in the document. The properties of a box specify how the contents of the box will be laid out — for example what its width is, and how the margins will be calculated. These properties are determined by the environment parameters of the box. Slugs are the subcomponents of boxes. In text they correspond to single lines, but in line drawings they may correspond to a component of the entire drawing. Slugs for text can have slug environments which allow changes in font and which perform underlining for example.

4.1 Box Environment Parameters

The attributes of a box determine the appearance of the box and the way in which the information in the box is laid out. The attributes are set to standard values on entering an environment, but they may be changed by using environment parameters, or globally by 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 4.2. In general the environment parameters do not affect the appearance of a box directly; instead they do so by being parameters to *computations*, which generate the values needed by the low level Mint formatting routines. Since the Mint user 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 that 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.

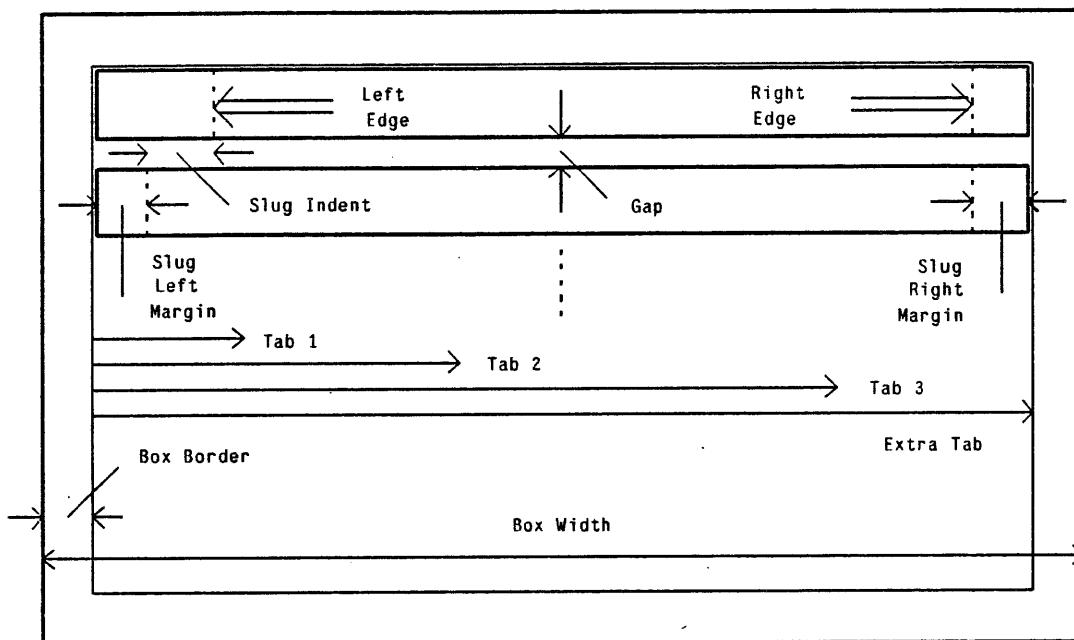


Figure 3. Box and slug parameters

4.1.1 Standard attributes

The following are the standard attributes, and the types of the values they may take. See section 2.3 for a description of these types, and how values of them are represented. <<Since the device characteristics are associated with the galley, one ought to be able to use page-relative units below. It's an oversight that you can't.>>

Above , Below

These specify the minimum space above and below a box (this statement applies to the standard environments; see section 4.4 for more details on this). The gap between two boxes, *A* followed by *B* is computed as the maximum of the *below* of *A* and the *above* of *B*. The values of *above* and *below* are specified in absolute units or lines, e.g.

```
@begin(Heading, Above = 3 lines, Below = 1.3 lines)
```

BackGroundColour

BackGroundColour determines the colour of the background of a box. See section 10.2 for a description of how to use this parameter.

Border

This determines the width of the border that exists between the outer edge of the box and the inner edge; see figure 3. It is specified in absolute units, (horizontal) ems or lines.

BorderStyle

The border style determines the appearance of the border drawn around the box. Two border styles are predefined — *NoBorder*, which leaves the border

	empty, and <code>Width1</code> which draws a border of width $1/100^{\text{th}}$ inch around the box. See section 10.1 for more details about creating border styles.
<code>CompLM</code>	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 provide his own, for all sorts of exotic purposes. The parameter takes small integer values. See section 4.4 for more details.
<code>CompRM</code>	In a similar way, this specifies the computation for the right margin.
<code>CompGap</code>	This specifies the computation for the gaps separating the slugs.
<code>CompXPosn</code>	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.
<code>CompYPosn</code>	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.
<code>CompWidth</code>	This specifies the computation to determine how the width of the box is derived.
<code>CompFont</code>	This specifies the font for each of the slugs in the box.
<code>Continue</code>	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 that box occurs; and if this parameter is <code>true</code> the margins are treated as if the box is a continuation of the previous box of this kind.
<code>ExtraLeftMargin</code>	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 absolute units or <code>ems</code> , e.g. <code>@begin(verse, extraleftmargin = 4 quads)</code>
<code>FaceCode</code>	This specifies the default face code of the box. It takes values from (<code>r, i, b, c, g, t, p, z, m0, m1, m2</code>) or from the face codes defined by the user. See sections 4.6.1 and 4.6.3 for the meaning of these values.
<code>FontSize</code>	This determines the default font size used for the fonts in the box. It takes values from (<code>11, 1, n, s, ss</code>). See section 4.6.2 for the meanings of these values.
<code>Gap</code>	This parameter specifies the gap that will be placed between the slugs in the box. See figure 3. It is specified in absolute units or <code>ems</code> (yes, <code>ems</code>).
<code>ImageColour</code>	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 10.2 for a description of how to use this parameter.
<code>JustifyLeft</code>	And <code>JustifyRight</code> , <code>JustifyLeftLast</code> , <code>JustifyRightLast</code> . These specify the nature of the justification actions taken when formatting text slugs.

JustifyLeftLast and **JustifyRightLast** control the justification of the last line of a box; **JustifyLeft** and **JustifyRight** control the justification of the other lines in the box. They take values from (**True**, **False**). Suitable combinations of these values are used to produce most of the different box formats; for example, if all are **false** the information in a box is centred, if **JustifyLeft** and **JustifyRight** only are **true**, then the information in the box is flushed left.

LeftMargin	This specifies the horizontal distance of the slug's left margin from the internal dimension of the box. The left edge of the slug's contents may be further indented by the value of Indent or ExtraLeftMargin , depending on other environment parameters. See figure 3. It is specified in absolute units or ems, e.g. <code>@begin(itemize, leftmargin = 5.5 ems)</code>
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 an absolute distance or one specified in lines, or it takes the value all , which is interpreted to mean that the box must not be split across pages.
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 (Skip , Default , TitlePage , Contents). See section 8.1 for more details on how choices of this parameter affect the page layout.
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 11.1.
RightMargin	This specifies the horizontal distance of the slug's right margin from the internal border of the box. See figure 3. It is specified in absolute units or in ems, e.g. <code>@begin(itemize, rightmargin = 20 picas)</code>
TabSet	This parameter takes an absolute distance or one specified in ems, and sets a tabulation at that point (measured from the exterior border 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, and there will be an additional tabulation set at the right edge of the inner border. See figure 3. Several values can be set by the same tabset . For example <code>@begin(example, tabset 5cm, 6.6666666cm, 8.3333333cm)</code>
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 TabClear should precede any uses of TabSet .
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. See figure 3. Any previous tabulations are destroyed.

UnderLine	This determines whether slugs in the box will be underlined, eraselined or overlined. 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>). Several non-conflicting values can be set at the same time.
Width	This defines the width of the external dimensions of the box that encloses the information in an environment. See figure 3. The width is specified in absolute units or ems, for example <code>@begin(plot, width 3.25 inches)</code> Note that the height of a box relative to its width is determined by its contents.

4.1.2 Additional parameters

Parameters additional to those above may be passed to certain environments. 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.

The following environments take additional parameters — the document type environments; several environments, such as `multiple` and `describe`, that allow others to nest within them; the `itemize` and `enumerate` environments; the `maths` environment; and the `TitlePage` environment.

4.1.2.1 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. These additional parameters have been designed to cause global changes in the “style” of the document, and for this reason they are called *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`. <<These attributes cannot be set to page-relative values, even though, anomalously, the `styles` object can be set in this way. This will get fixed.>>

BibStyle	This sets the citation style for the document. It takes the value <code>stdnumeric</code> , <code>stdalphanumeric</code> , <code>cacm</code> or <code>ieee</code> ; its default value is <code>stdnumeric</code> . See section 9.1 for a description of the bibliography facility, and a description of how to create new bibliographic styles.
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. If the string

	is <code>label</code> , then Mint will place the identifier of the label, converted to capital letters, in the place of the cross reference should it not be defined.
HGap	This sets the size of the “heading gap”, an extra gap used to compute the gaps for heading and section environments. It is specified in absolute units or ems.
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. See figure 3. It is specified in absolute units or ems.
IndexStyle	This sets the style in which the index will appear. It takes the value <code>macro</code> or <code>style1</code> ; the default is <code>style1</code> . See section 9.2 for a description of the index facility.
NAbove	This sets the size of the narrow above leading. It is specified in absolute units or lines.
NBelow	This sets the size of the narrow below leading. It is specified in absolute units or lines.
NGap	This sets the size of the narrow gap leading. It is specified in absolute units or ems.
NLM	This sets the size of the narrow left margin of the document. It is specified in absolute units or cms.
NRM	This sets the size of the narrow right margin of the document. It is specified in absolute units or ems.
RAbove	This sets the size of the regular above leading. It is specified in absolute units or lines.
RBelow	This sets the size of the regular below leading. It is specified in absolute units or lines.
RGap	This sets the size of the regular gap leading. It is specified in absolute units or ems.
RLM	This sets the size of the regular left margin of the document. It is specified in absolute units or ems.
RRM	This sets the size of the regular right margin of the document. It is specified in absolute units or ems.

4.1.2.2 Style parameters for other environments

In order to allow the user to impose stylistic requirements on portions of a document, several environments take style parameters. These environments are: `abstract`, `annotation`, `commentary`, `contents`, `describe`, `description`, `enumerate`, `figure`, `foot`, `itemize`, `multiple`, `notice`, `portion`, `researchcredit`, and `table`. Not all the style parameters are meaningful, though. In particular, `filler`, `indexstyle` and `bibstyle` are not allowed. The other style parameters have the same effects as described in the previous section.

4.1.2.3 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 8.2.4.2 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.

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 pageheight).
ResearchCredit	This specifies the position of the research credit box (0.80 pageheight).
CopyRightNotice	This specifies the position of the copyright notice box (0.75 pageheight).
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 pageheight).
Notice	This specifies where the (first or only) notice will appear (0.39 pageheight).
Notice1 ... Notice6	These specify where the corresponding notices will appear, counting in lexicographic order of appearance of the notices in the original manuscript.

4.1.2.4 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 itemize and enumerate environment. Its default value is 0.0625 of the width of the box.

The **itemize** environment also takes another parameter, **mark**, that allows you to specify the mark that will precede each of the items. The characters of the mark come from the font **MintFontn** which contains a collection of bullets and copyright symbols, and is otherwise empty. To use alter the mark you need to:

- Substitute the characters of the mark into **MintFontn**. Most positions are free, but to be safe, substitute for one of the ASCII printing characters. For example
 - `@SubstituteChar(PressFile, MintFontn, A, ZFont10, @char(#10))`

- Specify the `mark` environment parameter for the `itemize` environment. For example

```
@begin(itemize, mark A)
```

4.1.2.5 Maths

The `maths` environment takes a number of additional parameters. These are described in part 12.

4.2 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 when the environment is invoked.

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.

	EnvAbstract0	EnvAlign0	EnvAnnotation0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	RAbove, RBelow	NAbove, NBelow	NAbove, NBelow
Margins	RLM, RRM	RLM, RRM	RLM, RRM
Continue	Disallowed	Disallowed	Disallowed
Gap	NGap	NGap	NGap
Justifications	JL, JR, JLL, JRL	T, F, T, F	JL, JR, JLL, JRL
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Normal, Roman	Normal, Typewriter	Normal, Roman
Underlining	Off	Off	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	Box Y size	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	7	7	None

	EnvAppendix0	EnvAppendix1	EnvAppendixSec0
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	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	Extra large, Bold
Underlining	Off	Off	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	DefaultPS	Skip	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	3, 0, 0, 0, 0, 0	3, 0, 0, 0, 0, 0
Tabulations	7	7	7

	EnvAppendixSec1	EnvCaption0	EnvCentre0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	RA+3*RG, RB+RGap	NAbove, NBelow	NAbove, NBelow
Margins	RLM, RRM	RLM, RRM	RLM, RRM
Continue	Disallowed	Disallowed	Disallowed
Gap	RGap+HGap	NGap	NGap
Justifications	T, F, T, F	JL, JR, JLL, JRL	F, F, F, F
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Large, Bold	FS, FC	Normal, Roman
Underlining	Off	UnderLine	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	RAbove*4	Need	0
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	3, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	7	3	7

	EnvChapter0	EnvCommentary0	EnvCopyrightNO
Interpreter Width Above and Below Margins Continue Gap Justifications Image Colour Background Colour Font Underlining Raster Function Page Style Need Border size Border style Computations Tabulations	Text Width RA+4*RG, RB+2*RG RLM, RRM Disallowed RGap+2*HGap F, F, F, F ImageColour BackgroundColour Extra large, Bold Off RasterFunction DefaultPS RAbove*5 0 NoBorder 0, 0, 0, 0, 0, 0 7	Text Width RAbove, RBelow RLM + W8/4, RRM + W8/4 Disallowed NGap JL, JR, JLL, JRL ImageColour BackgroundColour Normal, Roman Off RasterFunction Skip Box Y size 0 NoBorder 0, 0, 0, 0, 0, 0 7	Text Width RAbove, RBelow RLM, RRM Disallowed NGap F, F, F, F ImageColour BackgroundColour Normal, Roman Off RasterFunction Skip Box Y size 0 NoBorder 0, 0, 0, 0, 0, 0 7

	EnvDefault0	EnvDescription0	EnvDisplay0
Interpreter Width Above and Below Margins Continue Gap Justifications Image Colour Background Colour Font Underlining Raster Function Page Style Need Border size Border style Computations Tabulations	Text Width RAbove, RBelow RLM, RRM Allowed RGap Justifications ImageColour BackgroundColour Normal, Roman Off RasterFunction Skip 0 0 NoBorder 1, 0, 0, 0, 0, 0 7	Text Width NAbove, NBelow 0, 0 Disallowed NGap JL, JR, JLL, JRL ImageColour BackgroundColour Normal, Roman Off RasterFunction Skip RAbove*2 0 NoBorder 0, 0, 0, 0, 0, 0 6	Text Width NAbove, NBelow NLM, NRM Disallowed NGap T, F, T, F ImageColour BackgroundColour Normal, Roman Off RasterFunction Skip 0 0 NoBorder 0, 0, 0, 0, 0, 0 7

	EnvDItem0	EnvDokument0	EnvDPO
Interpreter Width Above and Below Margins Continue Gap Justifications Image Colour Background Colour Font Underlining Raster Function Page Style Need Border size Border style Computations Tabulations	Text Width NAbove, NBelow RLM, RRM Disallowed NGap JL, JR, JLL, JRL ImageColour BackgroundColour Font size, Face code Underline RasterFunction Skip Need 0 NoBorder 4, 0, 0, 0, 0, 0 Tabs	Text Width RA, RBelow RLM, RRM Disallowed RGap JL, JR, JLL, JRL ImageColour BackgroundColour - UnderLine RasterFunction Skip 0 0 NoBorder 0, 0, 0, 0, 0, 0 None	DP Width RAbove, RBelow RLM, RRM Disallowed RGap JL, JR, JLL, JRL ImageColour BackgroundColour Normal, Roman Off RasterFunction Skip Box Y size 0 NoBorder 0, 0, 0, 0, 0, 0 None

	EnvEnumerate0	EnvExample0	EnvFigure0
Interpreter	Text	Text	Text
Width	Width	Width	Wd * 3/5
Above and Below	RAbove, RBelow	NAbove, NBelow	RAbove, RBelow
Margins	NLM+W8/3.5, NRM	NLM, NRM	RLM, RRM
Continue	Disallowed	Disallowed	Disallowed
Gap	0	NGap	NGap
Justifications	JL, JR, JLL, JRL	T, F, T, F	JL, JR, JLL, JRL
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Normal, Roman	Normal, Typewriter	FS, FC
Underlining	Off	Off	UnderLine
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	RAbove*2	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
Tabulations	0	7	3

	EnvFlushLeft0	EnvFlushRight0	EnvFoot0
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	F, T, F, T	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	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
Tabulations	7	7	None

	EnvFormat0	EnvGloss0	EnvHeading0
Interpreter	Text	Text	Text
Width	Width	W - W(siblings)	Width
Above and Below	NAbove, NBelow	NAbove, NBelow	RA+3*RG, RB+RG
Margins	RLM, RRM	RLM + W8/8, RRM	RLM, RRM
Continue	Disallowed	Disallowed	Disallowed
Gap	NGap	NGap	RGap+HGap
Justifications	T, F, T, F	JL, JR, JLL, JRL	F, F, F, F
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Normal, Roman	Succ (FS), FC	Large, Bold
Underlining	Off	UnderLine	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	0	Need	RAbove*4
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
Tabulations	7	3	7

	EnvItem0	EnvItemize0	EnvMajorHeading0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	NAbove, NBelow	RAbove, RBelow	RA+4*RG, RB+2*RG
Margins	0, 0	NLM+W8/3.5, NRM	RLM, RRM
Continue	Disallowd	Disallowd	Disallowd
Gap	NGap	0	RGap+2*HGap
Justifications	JL, JR, JLL, JRL	JL, JR, JLL, JRL	F, F, F, F
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Font size, face code	Normal, Roman	Extra large, Bold
Underlining	UnderLine	Off	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	Need	RAbove*2	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	0	7

	EnvMAnnotation0	EnvMFoot0	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	Disallowd	Disallowd	Disallowd
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

	EnvMultiple0	EnvNotice0	EnvPageCommand0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	RAbove, RBelow	RA, RBelow	NAbove, NBelow
Margins	0, 0	RLM, RRM	RLM, RRM
Continue	EnvContinue	Disallowd	Disallowd
Gap	RGap	NGap	NGap
Justifications	JL, JR, JLL, JRL	JL, JR, JLL, JRL	T, F, T, F
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	FS, FC	Normal, Roman	Normal, Roman
Underlining	UnderLine	Off	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	Need	Box Y size	RAbove*2
Border size	Border	0	0
Border style	BStyle	NoBorder	NoBorder
Computations	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	Tabs	7	7

	EnvPageHeading0	EnvPageFooting0	EnvPageOffset0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	Above,Below, 0, 0	0, 0	Above,Below, 0, 0
Margins	RLM, RRM	RLM, RRM	RLM, RRM
Continue	Disallowd	Disallowd	Disallowd
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	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

	EnvParagraph0	EnvPlot0	EnvPrefaceSec0
Interpreter	Text	Plot	Text
Width	Width	Width	Width
Above and Below	RA+RGap, RBelow	RAbove, RBelow	RA+4*RG, RB+2*RG
Margins	RLM, RRM	RLM, RRM	RLM, RRM
Continue	Disallowd	Disallowd	Disallowd
Gap	RGap-HGap	RGap	RGap+2*HGap
Justifications	T, F, T, F	JL, JR, JLL, JRL	F, F, F, F
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Normal, Bold	Normal, Roman	Extra large, Bold
Underlining	Off	Off	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	DefaultPS
Need	RAbove*4	Box Y size	RAbove*5
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	3, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	7	None	7

	EnvProgramEx0	EnvQuotation0	EnvResearchCr0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	NAbove, NBelow	NAbove, NBelow	RA, RBBelow
Margins	NLM, NRM	NLM, NRM	RLM, RRM
Continue	Disallowd	Allowed	Disallowd
Gap	NGap	NGap	NGap
Justifications	T, F, T, F	Justifications	JL, JR, JLL, JRL
Image Colour	ImageColour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour	BackgroundColour
Font	Normal, Algol	Normal, Roman	Normal, Roman
Underlining	Off	Off	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	Box Y size	0	Box Y size
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	0, 0, 0, 0, 0, 0	1, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0
Tabulations	7	7	7

	EnvSection0	EnvSubHeading0	EnvSubSection0
Interpreter	Text	Text	Text
Width	Width	Width	Width
Above and Below	RA+4*RG, RB+2*RG	RA+RG, RBelow	RA+3*RG, RB+RG
Margins	RLM, RRM	RLM, RRM	RLM, RRM
Continue	Disallowd	Disallowd	Disallowd
Gap	RGap+2*HGap	RGap-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	Normal, Bold	Large, Bold
Underlining	Off	Off	Off
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	Skip
Need	RAbove*5	RAbove*4	RAbove*4
Border size	0	0	0
Border style	NoBorder	NoBorder	NoBorder
Computations	3, 0, 0, 0, 0	0, 0, 0, 0, 0	3, 0, 0, 0, 0
Tabulations	7	7	7

	EnvTextPart0	EnvTitleBox0	EnvTitlePage0
Interpreter	Text	Text	Text
Width	Wd * 3/5	Width	Width
Above and Below	NAbove, NBelow	RAbove, RBelow	RAbove, RBelow
Margins	RLM, RRM + W8/8	RLM, RRM	RLM, RRM
Continue	Disallowd	Disallowd	Disallowd
Gap	NGap	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	FS, FC	Normal, Roman	FS, FC
Underlining	UnderLine	Off	UnderLine
Raster Function	RasterFunction	RasterFunction	RasterFunction
Page Style	Skip	Skip	TitlePS
Need	Need	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
Tabulations	3	None	none

	EnvVerbatim0	EnvVerse0
Interpreter	Text	Text
Width	Width	Width
Above and Below	NAbove, NBelow	NAbove, NBelow
Margins	RLM, RRM	RLM, RRM
Continue	Disallowd	Disallowd
Gap	NGap	NGap
Justifications	T, F, T, F	T, F, T, F
Image Colour	ImageColour	ImageColour
Background Colour	BackgroundColour	BackgroundColour
Font	Normal, Typewriter	Normal, Roman
Underlining	Off	Off
Raster Function	RasterFunction	RasterFunction
Page Style	Skip	Skip
Need	0	0
Border size	0	0
Border style	NoBorder	NoBorder
Computations	0, 0, 0, 0, 0	2, 0, 0, 0, 0
Tabulations	7	7

4.3 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 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 describe these conventions⁶. 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.

BoxCaption0	This environment is very similar to the <code>BoxSectionEnv0</code> environment, differing only in that 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.
BoxCommentary0	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 the gloss in a box adjacent to the first environment.
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 <code>Foot</code> , <code>Annotate</code> and the page heading and footing environments.
BoxCRTerm0	This is a specialized procedure used by the <code>PageCommand</code> environment. It sets up a cross reference before scanning in its body in a manner similar to <code>BoxStandard0</code> .

⁶ 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.

BoxDescribe0	This is a generalisation of BoxCommentary0 . 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 description environment and the commentary environment.
BoxEnumerate0	This procedure is very similar to BoxItemize0 , 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.
BoxFigure0	This procedure repeatedly accepts the bodies of the figures, and places a caption below each figure if there is such an environment in the .Mss file. Thus several figures can be collected together into one box.
BoxGalley0	This is the driving procedure for each of the galleys. Basically, it sits and loops over other non-terminal 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.
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.
BoxMaths0	This environment is similar to the BoxStandard0 , but it invokes some special processing to handle the maths environment.
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).
BoxPortion0	This routine is used in the separate formatting of documents. It is similar to BoxMultiple0 , except that it invokes some additional routines needed by the separate formatting facilities.
BoxRoot0	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.
BoxSectionEnv0	This procedure is similar to BoxStandard0 , except that it injects a prefix for the environment into the first slug that is made.
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.

BoxTable0

This procedure repeatedly accepts the bodies of the tables, and places a caption above each table if there is such an environment in the .Mss file. Thus several tables can be collected together into one box.

4.4 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.

4.4.1 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 than or equal to 8 are used to specify arbitrary computations.

CompFont

Only one computation is provided: 0. This sets the font to the font in the environment parameters.

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.
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.
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 1 places the box directly under the previous box.

4.4.2 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, Mint does 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

```
@setcompxx (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.

4.5 Miscellaneous layout statements

This section describes several layout commands: vertical and horizontal spacings, and tabulations. Two environments that are useful for laying out tabular information are also described.

4.5.1 Spacing statements

hsp

This takes a single parameter, which is a horizontal distance. If the statement occurs in an environment other than `pageoffset` 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.

vsp

This takes a single parameter, which is a vertical distance. If the statement occurs in an environment other than `pageoffset` 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.

newline	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. You can obtain the effect of <code>@newline(1)</code> with <code>@*</code> .
newpage	This is a macro that calls <code>zsp</code> , which is the basic statement controlling the generation of new pages. <code>Newpage</code> is defined in section 6.2.2.

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).

If you find yourself making extensive use of these statements you are probably misusing Mint. Think how to describe the effects you want with environments and page layouts instead.

4.5.2 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 allows is not permitted.

To clear the standard tabulations of an environment, the `TabClear` environment parameter is used. Tabulations are set using `TabSet` followed by several horizontal distances. Alternatively tabulations can be set equidistant along the box by `TabDivide`, which clears all previous tabulations. The tabulations are set from the inner border of the box. 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, 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 in the box parameters for an environment. Any number of additional tabulations can be set using `@↑`.

4.5.3 The Align environment

The `align` and `maths` environments treat `@\` differently from the way in which other environments treat it. (For more details about the `maths` environment see part 12.) 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,1,1,1,1)
@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 expense claimed)           @u(Amount claimed)

@w(Depreciation @\ - Fabric) $125@\n
@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@\.`

4.5.4 The `Describe`⁷ environment

Sometimes you need to place two or more environments side by side when, for example you are providing a commentary or gloss on a piece of text. The `commentary` environment allows you to do this kind of layout. In fact the `commentary` environment is a degenerate case of the `describe` environment, which allows several environments with different widths to be placed side by side. The default values of the

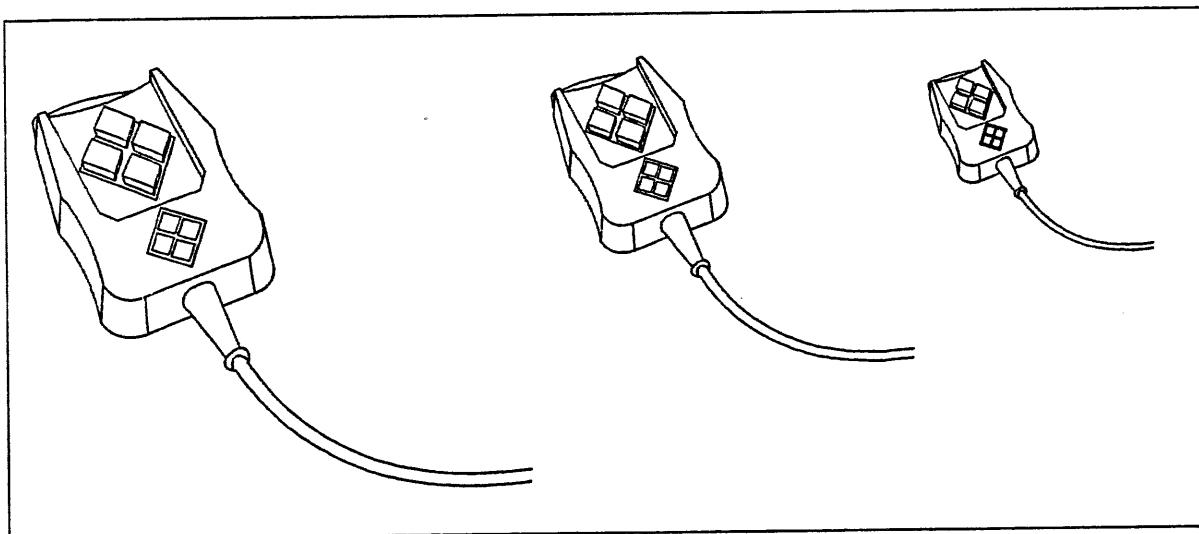
⁷ Pronounce it as des'cribe, or de'scribe, as you wish.

parameters that `describe` takes allow it to be used in a way similar to 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)
```

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 3.2.2 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 that at which items normally occur in a `description` environment.

If the boxes within the `describe` environment are not the same size in the *Y* direction, space is added to the bottom of the smaller boxes. Normally this space will not be visible; however, if borders are drawn round boxes, this rule will cause the borders to appear at the expected places.

4.6 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 that in which it starts.

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

4.6.1 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 5.3.3. (Standard typographical practice provides a set of small capitals for each font, in addition to the lower case and large upper case letters. ASCII does not recognize such niceties, however, and fonts designed for computer output have only lower case and large upper case letter sets. As a consequence it is necessary to use small capitals from a smaller font; this is not ideal because the weight of these small capitals is less than it would be if they were a part of the font. There is a trade-off, therefore, between the size of the small capitals and their weight. The small capitals used above are too large; if they were smaller they would be too light. Sigh! Alternatively, use the Metafont CMCSC font.)

<code>@g[greek]</code>	The <i>γρεεκ</i> type face.
<code>@t[typewriter]</code>	The <i>typewriter</i> face.
<code>@p[bold italic]</code>	The <i>bold italic</i> face.
<code>@z[symbols]</code>	A collection of mathematical $\sigma\psi\pi\beta\circ\lambda\sigma$.
<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 $\ -\sqcup/\sqleftarrow\{\nabla\Delta\sqleftarrow\in$ 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.

4.6.2 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.
<code>@ss[extra small]</code>	An extra small font.

If you need to change the font size and the face code simultaneously, you can combine the two slug environments. For example, `@s(@i(small italic))` can be written as `@si(small italic)`. All combinations of font sizes and face codes are allowed.

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 3.1.2), then Mint will supply some default font.

4.6.3 User Face Codes

Sometimes the face codes above are not adequate — you may want to use sans serif bold headings, or old english, or bold greek in your document. Mint allows you to declare new face codes using the statement `newfacecode`. A face code that has been declared can then be used to indicate slug environments, or as an argument to the `facecode` parameter of a box. For example, assume that you are preparing a text in which technical terms are to appear in an italic sans serif font. You can declare a new face code `iss` by writing

```
@NewFaceCode (iss)
```

after which you can write in your document

```
The elaboration of a @iss(label-definition) involves no action and  
yields no value.
```

You still need to decide which fonts will be used in the `iss` environment. This is done using `assocfont`, described in section 5.2.2.

The previous version of Mint allocated user face codes `f0`, ..., `f9`. The library `userfacecodes` contains the `newfacecode` definitions that declare these face codes, in case you do not want to change your manuscript. See section 14.2 for a description of the library facility.

4.6.4 Underlines, Overlines and Eraselines

The following are available:

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

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

4.6.5 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 already in the box at the position at which 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 11.1) which RasterOps are available. The following are available in Mint.

RRp1	Replace the destination by the source.
RNot	Replace the destination by the inverted pixels of the source.
RAnd	Replace the destination by the conjunction of the destination and the source.
RAndNot	Replace the destination by the conjunction of the destination and the inverted source.
ROr	Replace the destination by the disjunction of the destination and the source.
ROrNot	Replace the destination by the disjunction of the destination and the inverted pixels of the source.
RXOr	Replace the destination by the exclusive or of the destination and the source.
RXNor	Replace the destination by the exclusive nor 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.

4.6.6 Scripting

The following scripting slug environments are available:

@+(phrase)	Super ^{scripts} the phrase.
@-(phrase)	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 user. In general you should use the **maths** environment for superscripting and subscripting. See part 12.

4.6.7 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_2^1 , though in general you should be using Mint's mathematical typesetting facilities (part 12). Ovp environments can be nested.

Part Five

Fonts

Mint allows you to create documents using a variety of fonts. Fonts are collected up into families, and the families are associated with galleys. It is possible to create new font families, change which fonts are in the family, and alter which font family is associated with a galley. This section describes how to do this, and how to tailor fonts by selectively replacing some characters by characters from other fonts.

5.1 Font representations

Mint can handle several different font representations (simultaneously if needed — this occurs when cross-proofing, or when a document is being sent to several galleys, each with a different associated device). The understanding of a font representation has to be built into Mint, and at the moment it can handle two representations — `XeroxFormat`, used for press files, in which font information is extracted from a `Fonts.Width` file; and `KstFormat`, used for the Perq fonts, in which the font information is extracted from a file with extension `.Kst`. However, because all the information that Mint needs can be fed to it using Mint statements, it is also able to handle fonts for which no other external information is provided; also, since Mint is able to convert the information into a binary form, no penalty is imposed if this is done. Section 11.1.0.2 contains more details.

5.2 Font families

A galley accesses a font by indexing into the *font family* associated with the galley when it is created. A font family is a rectangular array of *font indicators* indexed by *font size* and *face code*; see sections 4.6.1 and 4.6.2 for more information about these terms. A font indicator must be written into the appropriate position in the font family for the galley to be able to use the given font size and face code. The standard font families have several font indicators already present; see section 3.2.2 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.

5.2.1 Creating font families

A font family is an object that can be created within Mint, loaded with font indicators, and associated with galleys. For example the statement

```
@NewFontFamily (ContentsFF, PressFile)
```

creates a new font family named **ContentsFF** which will be used to hold font indicators suitable for **pressfile** devices; all the elements of the array are initialized to a distinct value that allows Mint to recognize there is no font there. See section 11.1 for a description of devices and device classes. The next section describes how to associate fonts with the family.

A font family is associated with a galley when the galley is created: the **newgalley** statement takes a font family name as one of its parameters, see section 3.1.5. The font family associated with a galley can be changed at any time by the statement **SetFontF**, which takes a galley identifier and a font family identifier:

```
@SetFontF (Contents, ContentsFF)
```

will cause the contents galley, when it is activated, to use the fonts in the **contentsff** family.

The same font family can be associated with several galleys; changes in the font family will then be available to all the galleys.

5.2.2 Associating fonts with a font family

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

```
@assocfont(mainff, 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, the font is loaded into memory. Note that Mint uses the device class that has been associated with the family 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 class identifier to disambiguate the font identifiers (just as you would expect).

5.3 Logical 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). In addition, you may want some sequence of characters in the manuscript to be replaced by a single character — for example the sequence `f i` to be replaced by `fi`, or the sequence `a"` by `ä`.

The appropriate way of regarding this is to draw a distinction between *physical fonts*, which are provided to drive some device, and which supply some specific mapping between ASCII character codes and glyphs, and *logical fonts*, which provide an arbitrary mapping between character codes and glyphs. Mint deals with logical fonts, so you have control over what the mapping will be. Normally you will be happy with the mapping chosen by the font designer, and that is the mapping that Mint uses unless you tell it differently; occasionally you will want to change the mapping. In this section I describe the mechanisms that are used to alter the mappings, and to create new logical fonts.

5.3.1 Changing font mappings

Mint provides a general mechanism for replacing a sequence of characters in one font by a character in another, for replacing a sequence of characters by a blankspace of some user-specified size, and for replacing a sequence of characters 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 sequence of characters can be substituted by the statement `substitutechar`
`@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 `SubstituteCharR` 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,
`@substitutecharr(Dover, TimesRoman10i, A, Z, TimesRoman10b, a)`
will cause all upper case letters in TimesRoman10i to appear as lowercase bold letters.

- ◆ A sequence of characters 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 sequence of characters. This feature is of use for creating narrow gaps in fixed width fonts, for example.

```
@SubstituteGap (Perq, Gacha9, @char(sp), .5 quad)
```

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

- ◆ A sequence of characters can be replaced by an icon by the statement **SubstituteIcon**. Icons are dealt with in the next section.

When Mint is looking for a glyph it applies the transformations implied by the substitute statements repeatedly until they yield a printable object or a gap. (It doesn't bother to test whether this procedure terminates.) There can be several ways to specify some substitutions. For example, to specify that the ligature **ff i** should be replaced by **ffi** can be done in two ways: either

```
@substitutechar (pressfile, timesroman10, ff, timesroman10, @char(#06))
@substitutechar (pressfile, timesroman10, ffi, timesroman10, @char(#21))
```

or

```
@substitutechar (pressfile, timesroman10, ff, timesroman10, @char(#06))
@substitutechar (pressfile, timesroman10, @char(#06)i, timesroman10,
@char(#21))
```

5.3.2 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 desired width of the icon; the *X* and *Y* positions of the top left-hand corner of the bounding box, relative to the current position in the slug; and the distance from the current position to the next character after 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, 0.12in, 0in, 0.20in, 0.12in)
```

which specifies that 0 in **TimesRoman10** on the Dover will be replaced by the mouse icon. In terms of the quantities shown in figure 4, $XX - X0 = 0.12in$, $XRel = 0in$, $YRel = 0.20in$ and $XT = 0.12in$. I leave it as a simple exercise to work out how the tail of the mouse has been drawn.



5.3.3 New fonts

The statements **SubstituteChar**, **SubstituteGap** and **SubstituteIcon** (and **Kern** and **SubstituteInfo**, dealt with below) 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 (PressFile, 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. All the substitutions and kerns that have been made to **Saila10** before the **copyfont** statement will be copied into **mydirtyfont**.

The **c** slug environment has been created using

```
@copyfont(PressFile, CapsFont, TimesRoman10)
@substituterange(PressFile, CapsFont, a, z, TimesRoman8, A)
@assocfont(FONTs0, 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. To allow you to do this 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 *YY* and *Y0* given, using the conventions described in the next section. After it has been created, characters can be filled in using the substitute statements.

The statement

```
@EmptyFont (PressFile, MyFont, 11points, -1point)
```

creates a new 12 point empty font, with the top of the font bounding box 11 points above the origin, and with the baseline one point below the origin.

5.4 Character information

In order to place a character on the page, Mint needs to have several pieces of information. The minimum information it needs is the height of the top of the bounding box of the character above the baseline, the depth of the bounding box of the character below the baseline, and the width of the character. This minimum information is contained in the font representations; however, the quality of the typesetting that can be achieved by this minimum information is limited, and additional information is needed if better results are desired. Rather than require this information for all the fonts that it uses, Mint allows the information to be supplied optionally — if it is there, Mint will use it, and produce better quality typography; if it is not, Mint will fall back on using default information. The speed of Mint is not affected a great deal by making use of all the information (measurements show it runs about 5% slower if all the information about characters is given, than if only the minimum is given), however press files are much larger. The limitation on using the extra information is posed by the (personal) cost of extracting the information and representing it in a form that Mint can use. Normally you will not be concerned with what this information is, or how it is represented, since it will have been incorporated into the state files. You can regard this section as a collection of insights into the way Mint handles character information.

5.4.1 The values describing a glyph

The two drivers that Mint uses at present, the Perq driver and the Press driver, assume that each character is described by a *bounding box* and an *origin*. If you specify to the drivers that a sequence of

characters is to be placed side by side, then the driver will place the characters such that their origins are on the same horizontal line, and their bounding boxes are touching. This is a characteristic of the device that is understood by Mint. Should the information with which you describe each character correspond to Mint's understanding of the way the device works, then Mint will drive the device in a straightforward way. That is, if you need a sequence of characters on the page, and you have not specified that the origin is shifted for any of the characters, and you have not specified that there are to be gaps between the characters, then Mint will present a sequence of characters to the device and request it to output the sequence in the default way. If, however, you alter the character values then Mint may need to specify to the driver the position of each character.

Mint allows you to specify the size of the character bounding box and the position of the origin, in addition to several other values. Note however, that the size of the bounding box and the position of the origin are the basis of a contract between Mint and the driver: Mint assumes that whatever values these parameters have will indeed be the values that the driver uses, so if they are incorrect Mint will not be able to guarantee the appearance of the output. The possibility of setting these values has been provided to allow font characteristics to be defined entirely within Mint, without need to go to some other source of information (such as a fonts width file or a .TFM file). If you wish to cause the vertical or horizontal placing of characters to differ from the design placings, the correct way of doing this is to use other features of Mint, described below.

Each character is described by 16 values, which are related to the bounding box of the character and the bounding box of the ink of the character, and to two origins, as shown below.

The vector $X0, Y0$ measures the distance from the X origin to the bottom left corner of the bounding box of the character, and the vector XX, YY measures the distance to the top right corner of the bounding box of the character. Similarly the vectors $IX0, IY0$ and IXX, IYY measure the distances to the ink bounding box. The vector $VX0, VY0$ measures the distance from the Y origin to the bottom left corner of the bounding box. The distance XT measures the horizontal translation from the X origin of one character to the X origin of the next adjacent character; the distance YT measures the vertical translation when one character is placed on top of another (this normally only happens in mathematical layout). The value ICR measures the right hand italic correction, that is, the distance that the top right corner of the ink bounding box overhangs the bottom right hand corner; similarly the value ICL measures the distance that the bottom left hand corner protrudes to the left of the ink bounding box. Finally, the vector $XRel, YRel$ specifies the displacement from the position that the origin would normally be placed to where Mint instructs the device to place the origin. Nonzero values of $XRel$ shift the ink of the character in the horizontal axis, without affecting the position of adjacent characters; nonzero values of $YRel$ similarly shift it vertically.

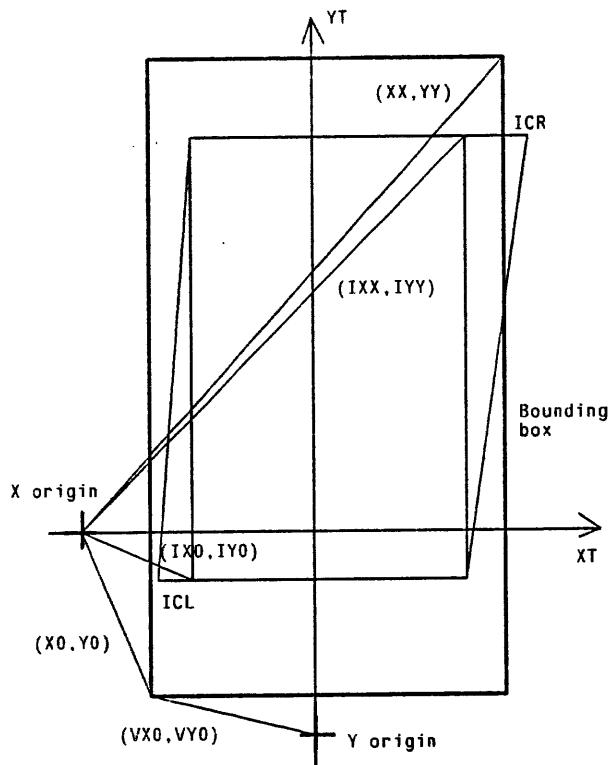


Figure 4. Character information

Since Mint can handle up to 128 fonts, each having up to 255 characters, and each character can have up to 16 values describing it, there are problems that have to be solved in order to store the information in a reasonable amount of space. Basically this is done by using several levels of indirection, which allows the same information to be used to describe several fonts and several characters. The way in which this information is stored and the way in which it is presented to Mint is beyond the scope of this section; section 14.3.4 gives more details.

5.4.2 Applying character information

Mint will automatically apply the character information when it is in the `\mathsf{maths}` box environment and the `\mathsf{m}` slug environment. If an item of information it needs has not been specified, it will use default information. The particular form these defaults take are too detailed to be included here, though they are just what you would expect.

In non-mathematical environments Mint will not normally use the information above; this is because press files are much larger when the information is used. In order to cause Mint to use the information you should use the statement `\substitute{info}`, which takes the form

```
@SubstituteInfo (PressFile, TimesRoman18, X)
```

This will cause Mint to use the character information for X in TimesRoman18 in Press Files, but otherwise to use the default information. The statement

```
@SubstituteInfoR (PressFile, TimesRoman18, a, z)
```

will cause Mint to use the information it has for all characters between a and z.

5.4.3 Extracting character information

Occasionally it is useful to extract information about a character from a font in a form that allows it to be used as a parameter to another Mint statement. For example, you might want to substitute a gap that is the same width as some character. The statement `CharInfo` extracts the information. For example

```
@CharInfo (Dev = PressFile, Font = TimesRoman10, C = @Char(Sp), Info = XT)
```

which will extract the information in the form 18 iu. The `info` parameter takes the values `x0, y0, xx, yy, ix0, iy0, ixx, iyy, vx0, vy0, xt, yt, icl, icr, xrel` and `yrel`.

5.5 Spacing adjustments

If characters are placed with their bounding boxes adjacent, the results are not always pleasing. Some characters appear too close together, and others too far apart. The reason for this is that the edges of the ink are not vertical, so that characters such as o next to x appear to have too much space separating them, whereas f next to ? have too little. By careful design of the font it is possible to reduce the effect, but in general the use of only first order information (the widths of individual characters) to determine positions is not sufficient, and you need both second order information (dependent on consecutive characters), and information about the *optical edge* of a character to get good layout. Traditionally this information has been captured in three ways: by *italic corrections*, which are used when an italic character is adjacent to a nonslanting character, by *kerning*, which is a second order adjustment of positions, and by *optical adjustments*, which are needed when a character is adjacent to white space, such as at the end or beginning of a line.

Mint allows all this information to be used. This section describes how it is specified, and its effect on the output.

5.5.1 Italic corrections

Left and right italic corrections are specified in the character information. The correction is applied if a character in an italic font is adjacent to a character in a non-italic font, this attribute being determined from the *slope* of the font, a part of the information that is associated with a font when character values are specified. Mint uses the height (or depth) of the characters to determine the amount of extra space to leave. For example, an italic *x* followed by a roman comma gets hardly any additional space, but an italic *y* followed by a roman brace gets the extra space specified for the *y*. This effect can be seen in the formula on page 137. The difference between text that is set with italic corrections and text that is set without can also be seen in the following. In (*fluff*) the parentheses intrude upon the *f* and *ff*, but with italic corrections in use the effect is (*fluff*). The amount of correction that Mint applies depends upon the relative slopes and sizes of the adjacent characters, so that if the roman characters are smaller, as are the full stops in *fluff*, they are placed closer to the italic characters, though still not as close as they would be if italic correction were not applied, when you would get *fluff*.⁸

Italic corrections are normally applied only in the mathematical environments. `SubstituteInfo` must be used to cause italic corrections in normal text.

5.5.2 Kerning

The *kern* of two characters is the extra space that is inserted between their bounding boxes; normally this space is negative. Since kerning adjustments can be made rapidly while scanning text, Mint does kerning in all contexts. A kern between two characters is specified by the `Kern` statement:

```
@Kern (PressFile, CMR10s10, o, -10 micas)
```

This will cause Mint to close the gap between *o* and *x* in font CMR10s10 by 10 micas.

Mint processes this statement by constructing a finite state recognizer; if there are several kerns and multicharacter substitutions rooted at the same character the finite state recognizer can become quite complex, but once constructed can operate quickly. For this reason it is advisable to put all the kerning statements into a definitions file so that they can be saved in a binary representation in the state file. Because of the way that Mint constructs the finite state recognizer, you have to specify all the substitutions that are rooted at a character before any of the kerns. Mint recognizes a kern after a sequence of substitutions in just the same way as a sequence of substitutions; that is, if you want to kern the ligature *fi* with a question mark, and also have *ffi* as a ligature, you write

⁸ Because the Dover driver has evolved piecewise, it produces more bulky output than it should. For this reason I elected not to use italic corrections regularly in this printing of the manual. However, by the time you read this the problem will have been fixed.

```
@substitutechar (pressfile, cmr10s10, ff, @char(#173))
@substitutechar (pressfile, cmr10s10, @char(#173)i, @char(#176))
@kern (pressfile, cmr10s10, @char(#173)?, 39 micas)
```

5.5.3 Optical adjustments

Mint does not yet perform optical adjustments. Watch this space.

Part Six

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. Mint's macrogenerator is modeled on that described by Strachey in Computer Journal, 1963, though many cosmetic changes have been made. In addition to 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, NewGalley etc; and finally it is used as an integral part of the Note facility, within which the bibliography facility acts as a subset.

6.1 Macro expansion

Let us start off with a few words of warning. First word of warning: macroexpansion is at best a feeble computational tool, which is made more dangerous because the parsing of a string that is to be macroexpanded can only be done incrementally, after the results of the macroexpansions in progress have been obtained. This is because a macro can generate a bracket, for example, that can alter the macrogenerator's scanning action. Those languages that keep a more decent distance between their data and their program text allow a formal evaluation of programs (corresponding to syntactic and semantic analysis) that is independent of the actual computation the program performs. If you infer from this that I believe macroexpansion is a mess, you are accurate. However, until I design a better front end you will have to live with its oddities. Second 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.

6.1.1 Input conventions

A macrogenerator statement has the following appearance:

```
@IsEq (abcd, efg)
```

with the command identifier preceded by @, and the parameters enclosed in brackets and separated by commas. In agreement with the conventions of the rest of Mint, 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.

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 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, and to balance the brackets of the (unevaluated) calls within the quoted string; 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.

In addition to providing 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 n^{th} actual parameter being bound to the n^{th} formal unless the actual parameter is named. 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 6.1.2 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 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 lexicographic 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.

6.1.2 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 6.1.3. (There are two other ways. `SpecialForm` is used to create state files, and is discussed in section 14.3.1.1. `EDef` can only be used in the `maths` environment, and is discussed in section 12.2.2.2.)

`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)
```

6.1.2.1 Accessing parameters

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)})
```

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 example 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.

6.1.2.2 Accessing 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 6.2.4 lists the values accessible in this way.

6.1.3 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 *Plagiary 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 *Plagiary List*. The *Plagiary 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.

6.2 Standard Macrogenerator Facilities

This section lists the predefined macros (actually those defined in the standard state files). 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 in detail are those termed *special macros*, which are concerned with macro expansion.

6.2.1 Special Macros

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

<code>andm</code>	<code><up to 16 parameters></code>
<code>brkt</code>	<code>b,a</code>
<code>char</code>	<code>ch</code>
<code>cond</code>	<code>if,then,else</code>
<code>defer</code>	<code>macro</code>
<code>form</code>	<code>id,params,body</code>
<code>iftrue</code>	<code>if,then</code>
<code>incptn</code>	<code>file</code>
<code>isdefined</code>	<code>param</code>

iseq	x,y
message	msg
notm	x
orm	<up to 16 parameters>
reinvoke	dm
value	id
w	word

Form, **Value**, **Defer**, and **ReInvoke** have been dealt with in sections 6.1.2 and 6.1.3. **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 ln for new line. 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. **IfTrue** returns its **then** parameter if its **if** parameter is a non-empty string. **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. **Incptn** is only used internally in the separate formatting facility.

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 anomalous effects on characters created by **Char**, and on certain strings created by **NConv**. That's a bug.

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

6.2.2 Extra macros

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

```
@Form (ref, lab, @""@nconv(placestyle,place,@value(lab)))
@Form (eqn, lab, @""@nconv(equationstyle,equation,@value(lab)))
@Form (pageno, lab, @""@nconv(pagestyle,pageno,@value(lab)))
@Form (figref, lab, @""@nconv(figurestyle,figureno,@value(lab)))
```

```
@Form (tabref, lab, @""@nconv(tablestyle,tableno,@value(lab)))  
@Form (newpage, n,  
@""@pagecommand(@zsp(@cond(@isdefined(n),+@value(n),+1))))"  
@Form (blankspace, n, @""@vsp(@value(n))")  
@Form (eval, x, @""@form(ev2,@value(x))@ev2())"  
@Form (portion, x,  
@""@brkt(@"[@begin{portion}],@"[@end{portion}])@incptn(@value(x))")
```

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

6.2.3 Summary of other macros

The following is a complete list of the statements accepted by the macrogenerator, together with their formal parameters, and the section in which they are described <<in the next edition, anyway>>.

addclass	nterm,rhs
adddefault	nterm,def
addpageinfo	pparams,area,backgroundcolour,border,borderstyle
alter	id,by
andm	<up to 16 parameters>
applictrans	cn,cv
assoccontents	id,env
assocconv	conv,nstyle
assoccref	e,c
assocfont	fontf,fontsize,facecode,fontname
assoclayout	pres,part,pstyle,layout
assocpart	pres,part
assocprefix	e,pi
assocprivf	fontf,type,fontname
assocproc	procf,environment,boxroutine,parameters,prefix
bibinclude	<up to 16 parameters>
bind	id,value,binding
bindcurval	id,binding
brkt	b,a
char	ch
charinfo	dev,font,c,info
cite	<up to 16 parameters>
citeincollection	collection,<up to 16 parameters>
comment	body
cond	if,then,else
copycharbbis	d,s
copycharbbrs	d,s
copycharbbxs	d,s
copycharbbys	d,s
copyfont	device,dfont,sfont
crossproofingdefault	targetdevice,viewingdevice,font
crossproofingfont	targetdevice,targetfont,viewingdevice,viewingfont
crsyntax	e,p
defer	macro
dumpstate	file
edef	id,body

```

emptyfont           device,dfont,height,baseline
form               id,params,body
iftrue              if,then
include             file
incptn             file
index               key,seckey,rest
indexinclude        <up to 16 parameters>
indexincollection collection,key,seckey,rest
initlexmap         m,c,e,v
isdefined          param
iseq               x,y
hsp                length
kern               device,dfont,dstring,sliver
label              id
library            file
make               documenttype,form=0,device=perq,rest
makecref           t,n,c
makeicon            slugtype,id,file
makerep             pres,pg,ag1,ag2,ag3,ag4,ag5,ag6
makeprefix          i,t,p1,p2,p3
makestyle           dev,id,wd,r1m,n1m,rrm,nrm,elm,rab,nab,rbe,nbe,rg,ng,hg,ri,
                   j1,jr,j1l,jrl,ic,bc,rf
mathsvca           vca,p,a,b,c,d,v0,v1,v2,v3,v4,v5,v6,v7,v8,v9
mathparams          f1,f2,f3,f4,p1,p2,p3,b1,b2,b3,b4,e1,g1,i1,i2
mdef               slex,dlex,stype,fcode
message             msg
moveto              dev,x,y
ncconv              conv,counter,label
newareaparams      id
newbib              id,st,sp,tm,sfn,vfn,ls
newbibcollection   collection
newborderstyle     id,mtl,mtr,mbr,mb1,patt,patr,patb,pat1
newcharbbis        id
newcharbbrs        id
newcharbbxs        id
newcharbbys        id
newcolour          colour,hue,saturation,brightness
newcontents         id,fe,xb,yb,ap,hf,exp
newcontentstable   id,cap
newcounter          id,within,start=1,change=+1
newdclass           id,dr,fm,xs,ys,is,rset
newdefault          id,fe,xb,yb,ap,hf,exp
newdevice            id,dc,xsz,ysz
newfacecode         fc
newfontfamily       id,dev
newfontvalues       id
newgalley           id,gt,ff,pf,s,e,a
newindexcollection collection
newline              lines
newpagestyle        id,cont
newpasteup          id,fe,xb,yb,ap,hf
newpattern          id,w0,s0,c0,w1,s1,c1,w2,s2,c2,w3,s3,c3,w4,s4,c4
newpcs              c
newpresentation    pres
newprocfamily       id
newproduction       id,c,d,de,pk,b1,ac,cr,h0,h1,h2,h3,h4,h5,h6,h7
newsyntaxclass     id

```

newtitlepage	id,fe,xb,yb,ap
next	id
notm	x
orm	<up to 16 parameters>
putchar	dev,c,font,x,y
putline	dev,x,y,w
readdefs	defs
readstate	file
register	c,o
registerenv	env
reinvoke	dm
remclass	nterm,rhs
remdefault	nterm,def
set	id,value
setcharbbi	id,ch,ix0,iy0,ixx,iyy,ic1,icr
setcharbbr	id,ch,xr,xt
setcharbbx	id,ch,x0,y0,xx,yy
setcharbby	id,ch,vx0,vy0,yr,yt
setcompfont	dev,cno,sno,val
setcompgap	cno,sno,val
setcompim	cno,sno,val
setcomprm	cno,sno,val
setdump	dc,fm,dv
setfontf	gid,ff
setfontinfo	dc,fi,wt,wd,ps,sl,fv,ew,eh,iv,xv,yv,rv
setfontvalue	id,c,e,v
setnext	id
setnotearea	g,a,ss,ns
setprocf	gid,pf
setscale	s
setsptype	left,right,spacing
setsvalue	spacing,fs,value
setstyle	gid,s
setunit	id,s
specialform	id,params,varp,no
stdconv	i,n
substitutechar	device,dfont,dstring,sfont,schar
substitutecharr	device,dfont,from,to,sfont,schar
substitutegap	device,dfont,dstring,size
substituteicon	device,dfont,dstring,icon,s,x,y,w
substituteinfo	device,dfont,dchar
substituteinfor	device,dfont,from,to
tag	id
value	id
vsp	length
w	word
within	file
zsp	page

6.2.4 System attributes accessed via @Value

The following values are available by using **Value**.

BibStyle	The current bibliography style.
Date	Yields the current date, as provided by the Perq. There are not yet any transformations on the format of the date.
Device	The device identifier.
DocType	The document type.
Form	The value passed to the <code>form</code> parameter of the <code>make</code> statement.
IndexStyle	The current index style.
Manuscript	The root file for the manuscript. The string returned by this parameter is the full path name, less the extension.
SourceFile	The current source file from which input is being taken. The string returned by this value is the full path name.
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.

Part Seven

Cross references

A technical document needs to have cross references between its parts. This is normally done by numbering sections, figures, etc, and using the numbers. You need then to have counters which can increment through the document, and you need to be able to refer to the values of the counters in a variety of styles. This section first introduces the notion of a counter, and describes the operations on it, then describes how you can control how the value of a counter appears in the document, and it finally describes the prefixes and postfixes which are attached to some environments.

7.1 Counters

In order to 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 therefore helpful to have each of these use distinct labelling schemes, so that one can refer to chapter one, figure VI, 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 styles. This section describes the facilities provided.

7.1.1 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 document⁹.

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.

7.1.2 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

⁹ 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 lines with counter-changing statements.

counters, the second parameter should be empty. Counters can start at any positive or negative integer value, and the increment can be any positive or negative integer value.

It is possible to set a counter to some arbitrary value 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 galley 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 which is described in section 7.3; in this case you should use the statement **SetNext**; for example

```
@setnext (authorcounter)
```

Unless you know what you are doing 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.

7.2 Labels

A label is defined by the statement **Label** (**Tag** may also be used); this takes a snapshot of the counter contour, and associates it with the label. The label identifier can be any valid macrogenerator string¹⁰. For example

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

¹⁰ Not, it seems, for labels defined in the **maths** box parameters. Sigh!

Labels may be defined in any of the galleys. When the galleys are created, the contour associated with a label is that appropriate for scanning the galleys in some canonical order (the order in which the galleys have been defined); when pages are created, 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.

7.2.1 Referring to labels

Mint 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.

Frequently you want to use the current value of a counter, rather than the value at some label. This is generally the case 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)
```

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.

7.2.2 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, Rest = @""Filler = ****")
```

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

When you are first creating a document, there are generally many unresolved labels. It is useful to know which ones they are, and where they occur in the manuscript. If you set **Filler** to the special value **Label**, then Mint will use a filler that comprises the label identifier in upper case.

7.3 Conversions

Mint provides a number of in-built *basic conversions*, such as **RomanUC**, a complete list of which is given in section 7.4.1. 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**.

7.4 Standard Conversions and Counters

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

7.4.1 Conversions

The following basic conversions are available.

RomanLC Converts a value into lower case Roman numerals, e.g. **vii**, **xxiv**.

RomanUC Converts a value into upper case Roman numerals, e.g. **MCMXLI**.

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. one hundred and three.
AlphaUC	Converts a number into cardinal English numbers in which the first letter of each word is a capital letter, e.g. Fifty Five.
LetterLC	Converts a number into a lower case letter, 1 = a, 2 = b, etc.
LetterUC	Converts a number into an upper case letter, 1 = A, 2 = B, etc.
Binding	Retrieves the arbitrary binding associated with the counter. The binding can be any string.

7.4.2 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 7.4.2. Each individual value is converted according to the style in the <code>nconv</code> statement in with 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 7.4.2, whereas I wanted subsection 7.4.2. I'll try again.
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 12.1.3.2 for more details.

7.4.3 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

7.4.4 Counters

The following counters are defined for use in the standard prefixes (section 7.5), the standard annotations (section 14.3.5.6), and in the standard presentation (section 8.3). The counter identifier, the counter within which it counts (if any), the initial value and the increment are shown.

7.4.4.1 Counters common to all document types

PartNo		1	+1
PageNo	PartNo	1	+1
FigureNo		1	+1
EquationNo		1	+1 ¹¹

7.4.4.2 Counters in document types with footnotes and annotations

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

7.4.4.3 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

7.4.4.4 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

¹¹ In document types with sections or chapters, the equationno counter counts within the corresponding counter.

7.5 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. Prefixes are specified along with the procedures and the environment parameters in the procedure family of a galley. The `maths` environment has a postfix string generated for it, but I'll call all inserted strings *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 compared with 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.

7.5.1 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, followed by an applied occurrence of the appropriate counter (<code>Section</code> , <code>SubSection</code> , <code>Paragraph</code>), using the <code>PlaceStyle</code> 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, 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, the word <code>Table</code> , followed

by an applied occurrence of the **Table** counter, using the **TableStyle** conversion style. The prefix string appears in the prevailing **b** 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, the word Chapter or the word Part , followed by an applied occurrence of the Chapter counter, using the ChapterStyle 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, the word Appendix , followed by an applied occurrence of the Appendix counter, using the AppendixStyle 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 Copyright , 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. (Later note: this is now done.) |
| Equation Postfix | The equation postfix is appended to the end of a formula created by the maths environment if it has a label parameter. It comprises an equation number (see section 12.1.3.2) enclosed within parentheses. |
| Editor Postfix | This postfix is used in the annotation feature, to distinguish the annotations of several editors. It will be described more fully later. |

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

PrefixChapter0	ChapterNo	A chapter prefix using Chapter
PrefixChapter1	ChapterNo	A chapter prefix using Part
PrefixAppendix0	AppendixNo	A appendix prefix
PrefixSection0	SectionNo	A place prefix
PrefixSubSection0	SubSectionNo	A place prefix
PrefixParagraph0	ParagraphNo	A place prefix
PrefixAppendix1	AppendixNo	A place prefix
PrefixAppendixSec0	AppendixSecNo	A place prefix
PrefixFigure0	FigureNo	A figure prefix
PrefixTable0	TableNo	A table prefix
PrefixCopyrtNo	CopyrtNo	A copyright prefix
PostfixEqn0	Equation	An equation postfix

Part Eight

Page layout

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. In this section presentations and layout routines, which create pages, are described.

8.1 Presentations

Mint allows several different presentations to be created from the same galley. Each presentation corresponds to the application of some set of page design rules, which are algorithmically described by layout routines. 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 galley will contribute to a presentation, so that this loss of flexibility may be disguised. It is possible, for example, to have two separate `Main` galley, 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 galley that contribute to a presentation must use the same target device.

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

8.1.1 The structure of a presentation

A presentation has two components: first, a vector that maps between the *page styles* that are specified in the box environment parameters, and *page layouts*, which are collections of formatting rules that lay out pages; and second, a collection of *document parts*, which are the pages which have been produced using the particular layout rules. To obtain any particular page layout, you specify the appropriate page style in the

box environment parameters. For example, the `TitlePage` environment specifies the page style 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.

Each collection of page layout rules is associated with a part, which receives the pages as they are created. Pages are placed consecutively in the part by order of arrival. However, the *printing order* of the pages in a presentation is quite arbitrary, since all the pages are created before any printing is performed.

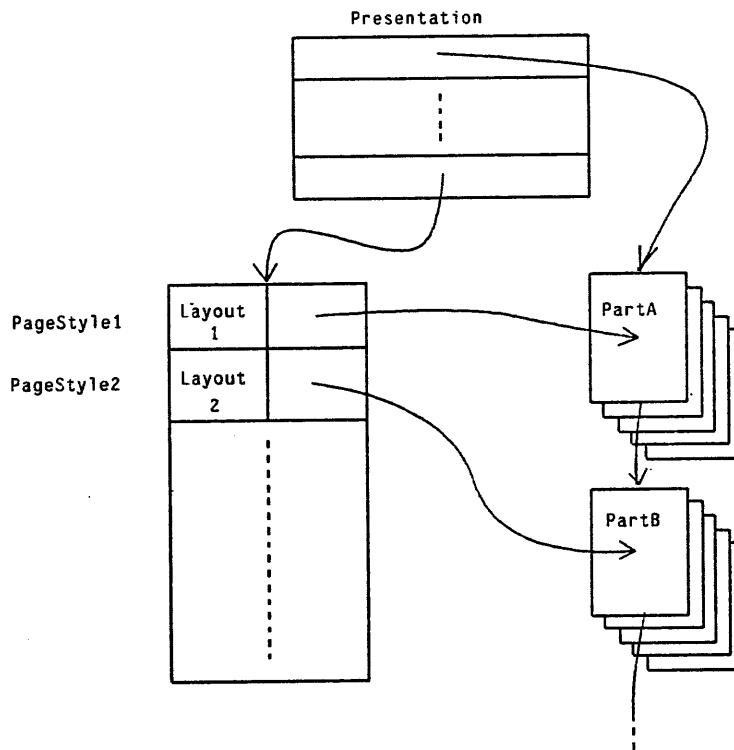


Figure 5. Structure of a presentation

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 8.2.2.2.

8.1.2 Page styles

Any particular document may need several different page styles. For example, the title page will differ from the layout of the main text of the document, which may be single column, and that may differ from the index which could be laid out using two columns. More elaborate document designs may require more page styles. Mint provides for this requirement by allowing you to declare *page styles*, and then to specify which boxes are to be laid out according to that page style. For example

```
@NewPageStyle (Acknowledgements, True)
...
@begin(heading, pagestyle=acknowledgements)
Acknowledgements
@end(heading)
I wish to thank all those battle-scarred pioneers who helped me with
invaluable comments
during the writing of this system.
```

declares a new page style, and then associates it with the heading. The page style is ignored during galley creation, but it is carried through to the presentation, where it is used to select one of the layout routines, and parametrize it with suitable parameters for laying out an acknowledgements page. (The layout routine might well be the same one as is used for title pages, but with different parameters.)

The second parameter to the `newpagestyle` statement specifies whether the page style is *continuing*. A continuing page style causes all the boxes that follow it to be treated using the same page style, unless they explicitly specify some other page style¹².

Several page styles are already declared; they are described in section 8.3.1.

8.1.3 Layout routines

Layout routines are built in to Mint. Each layout routine has an identifier; currently Mint has four layout routines: `TitlePage` which understands how to lay out title pages and other single page displays, `Contents` which helps lay out tables of contents, `Pasteup` which allows you to specify the appearance of pages interactively, and `Default` which lays out everything else. More details of these are given in section 8.2. Soon there will be layout routines for letters, multi-column pages, etc.

From a layout routine you can create a *layout*: this is an association of a layout routine and a collection of parameters that will get passed to the routine when it is called to lay out some pages. It is these layouts that are associated with page styles in the presentation mapping vector. Each layout routine uses different

¹² This isn't quite true, but it's a good approximation.

parameters, so it is more appropriate to describe them separately in section 8.2.3; the statements that create layouts all have a similar form though. For example

```
@NewTitlePage (TLayout, True, . , )
```

creates a layout that will result in the `TitlePage` routine being called with the four parameters specified (the last three of which will be whatever the default values are).

8.1.4 Defining new presentations

The `NewPresentation` statement creates a new presentation, and sets the elements of its layout mapping vector to undefined values. Empty parts are created by the `AssocPart` statement. For example,

```
@NewPresentation (Standard)
@AssocPart (Standard, TitlePage)
@AssocPart (Standard, MainBody)
```

creates a presentation named `Standard`, and associates two parts with it.

8.1.5 Making representations

A *representation* of the manuscript is 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.

Mint currently creates a single representation after the galleys have been created. By default this representation uses a presentation called `standard` that is automatically declared, and uses the `main` galley and the `footnotes` galley. A table of contents is created if the `contents` galley exists (see section 9.3). If you want a different presentation you can use the `makeRep` statement. This takes a presentation, a principal galley, and up to six associated galleys. For example

```
@MakeRep (Standard, Main, FootNotes, Annotations)
```

will create a representation with both footnotes and annotations (which will come out looking like

footnotes). This statement can occur at any point in the manuscript. Mint will store its parameters, and apply them when the galleys are complete.

8.1.6 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 11.2.

The following illustrate the methods of causing printing to occur.

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

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

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

will print pages 1 to 10, and

```
@PrintPresentation (Standard, Dover)
```

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 after it has formatted a document.)

8.2 Layout routines

Layout routines 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 routines 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 routines.

8.2.1 Sorting the slugs and boxes

The galley has 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 galley 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 — they are not in the same slug.

To assist the page layout routines a second structure is created which 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. Several collections of slugs are then created from the sorted list. 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).

8.2.2 Page areas

Each layout procedure regards the page as being divided into a number of nested *areas*. The number of areas, and the way they nest is determined by the layout routine itself; however the sizes, positions, border widths, border styles and background colours of these areas are determined by the parameters that are passed to the layout routines. Since the number of areas in a page can be large (the `Default` routine uses nine areas per page), it is not convenient to specify all the parameters explicitly. Instead Mint assumes that there are default values for the parameters, and that a list of modifications is passed to the routine. The manner in which this is done is somewhat arcane, but changing the area parameters is not done by the casual user.

8.2.2.1 Area parameter objects

It is possible to change the border width, border style, and background colour of each of the areas for each of the layout routines (see part 10). The parameters affect all the pages that are made by the routine.

First, an *area parameter object* must be created. This is done with the statement `NewAreaParams`; for example

```
@NewAreaParams(ColouredSlides)
```

After an areas parameter object has been declared, any number of modifications can be attached to it. They form a list which will get processed by the layout routine to which the list is passed in the order in which the modifications have been attached. Modifications are attached by the statement **addareainfo**:

```
@AddAreaInfo (ColouredSlides, Main, Yellow, 0.05cm, Width2)
```

which will cause the layout routine to which **ColouredSlides** is passed to colour the **Main** area of each page **yellow**, and to leave a border of 0.05cm, with a **width2** border style.

Since the way that the areas parameter object is passed differs for each layout routine, sample calls will be shown later.

8.2.2.2 Standard 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 is 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 **PH** and **PW** are the page height and page width; they are derived from the device characteristics (section 11.1). The values **BH** and **BW** specify the page border sizes in the vertical and horizontal directions; they can be set when a layout is created, as described below. If they are not set, they take the values of one eighth of the page height and one eighth of the page width. The border widths, border styles and background colours can also be set when a layout is created; their default values are **0**, **noborder** and **transparent**.

Default, Contents, Pasteup

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

Middle	FootNote	NoArea	0	0	PW-2*BW	PH-2*BH
FootNote					PH-2*BH	PW-2*BW
Middle	NoArea	NoArea	0			0

TitlePage

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

8.2.3 Creating layouts

A separate Mint statement exists for each layout routine. The Mint statement takes several parameters and parcels them up into a layout object that can then be associated with an entry in the layout vector. The statements are:

- | | |
|---------------------|---|
| NewTitlePage | The first parameter is the identifier of the layout, the second a boolean specifying whether a blank page is to be produced to make the number of pages created by the layout an even number. The third and fourth parameters are the size of the <i>X</i> and <i>Y</i> border widths (if omitted, the defaults are used), and the final parameter is the area parameters object, which may be omitted. |
| NewPasteup | This takes six parameters. The first five parameters are the same as those for NewTitlePage , and the sixth parameter specifies whether a heading is to appear on the first page. |
| NewDefault | This takes seven parameters. The first six parameters are the same as those for NewPasteup , and the seventh parameter specifies whether the gap between the slugs is to be increased to fill the <code>main</code> area. This feature is not yet implemented. |
| NewContents | This takes seven parameters. They are the same as those taken by NewDefault . |

8.2.4 Actions of the layout procedures

8.2.4.1 The Default layout routine

The routine 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` is placed into the `Heading` area and the next box from the `PageFooting` is placed in the `Footing` area, except on the first page if `HeadingFirst` is false, when only the page footing box is used. 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 on the even numbered page. I ought to fix that.

8.2.4.2 The TitlePage layout routine

This routine produces a page laid out according to information that is passed to it 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 `TitlePage` procedure. The values can be changed by passing parameters to the environment. See section 4.1.2.3 for details.

8.2.4.3 The Contents layout routine

This routine is used to lay out the table of contents; it is parasitic on the layout routine in the `Default` position of the layout vector, which it calls when it has loaded up the `Contents` galley with information extracted from the representation. More details about tables of contents are given in section 9.3.

8.2.4.4 The Pasteup layout routine

This routine interacts with the user, to allow page layouts that are not easily described algorithmically. This layout routine will be of particular value in creating slides; it is not useful for large documents that have a regular page structure, because the appearance of a page can be easily destroyed if the box spacings are irregular.

The routine allows boxes and slugs to be selected from any of the galleys, to be rescaled to be wider or narrower (in fact any of the box environment parameters can be changed), to be sliced into parts, and to be moved on the page. I won't describe the routine further, since there are some problems it has with slugs that have cross references, but I'm happy to give a tutorial to anyone who wants to use it.

8.3 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.

8.3.1 Page styles

The following are the standard page styles.

```
@NewPageStyle (, False)
@NewPageStyle (Skip, True)
@NewPageStyle (Default, True)
@NewPageStyle (TitlePage, False)
@NewPageStyle (Contents, False)
```

A description of these page styles is given in section 14.3.6.1. Since anyone declaring page styles must be an expert, they will certainly be familiar with that section.

8.3.2 Standard presentations

The following layout routines are specified. Their actions are described in section 8.2.

Default Contents TitlePage Pastup

The following is the standard presentation.

```
@NewPresentation (Standard)
```

```
@AssocPart (Standard, TitlePage)
@AssocPart (Standard, Contents)
@AssocPart (Standard, MainBody)

@NewTitlePage (TLayout, True, . . . )
@NewContents (CLayout, True, . . . , False, False)
@NewDefault (DLayout, True, . . . , False, False)
@NewPasteup (PLayout, True, . . . , False)

@AssocLayout (Standard, TitlePage, TitlePage, TLayout)
@AssocLayout (Standard, Contents, Contents, CLayout)
@AssocLayout (Standard, MainBody, Default, DLayout)
```

The table of contents is created only if there is an entry in the `Contents` 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 `Default` entry of the vector. See section 9.3 for more details about tables of contents.

8.3.3 Printing the standard presentation

If there has not been any `makerep` statement in the manuscript, Mint will create the following representation after it has created the galleys.

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

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

8.4 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.

8.4.1 Page headings and footings

The `PageHeading` and `PageFootting` 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 `Default` 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 **PageFootting** 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

```
@pagefootting(@pageno())
```

suffices.

8.4.2 Page offsets

The **PageOffset** environment feeds slugs to the layout procedures in a manner similar to that for **PageHeading** and **PageFootting**. 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)
```

8.4.3 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. Soon it will be possible to pass other information, such as colour and border style, to the layout routines, to allow the appearance of individual pages to be controlled.

8.4.4 Example of headings and footings

A comprehensive example of the use of `PageHeading` and `PageFootting` environments is provided by the macro that is used for the heading and footing of this manual. It is

```
@form(chpgno,,@""@nconv(arabic,chapterno,)--@pageno())
@form (chap,@"(chapid,authid,docid,lab),@""@Chapter(@value(chapid))
@iftrue{@isdefined(lab),@label(@value(lab))}
@begin(pageheading)
@begin(align,border=0.1in,borderstyle=width1,tabdivide 2)
@r(@chpgno()) @>@b(@value(chapid))@< @ovp( @+(@w(@value(authid)))@\)@-
@w(@value(docid)))@\
@end(align)
@begin(align,border=0.1in,borderstyle=width1,tabdivide 2)
@ovp(@+(@w(@value(authid))))@-(@w(@value(docid)))
@>@b(@value(chapid))@< @r(@chpgno())@\)
@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)
")
```


Part Nine

Documentation aids

Technical documents generally contain citations to other works, and are supported by tables of contents and indexes. Mint supplies facilities for generating bibliographies, for creating tables of contents and for managing indexes; they are described in this section.

9.1 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 than Scribe to handle unusual reference formats, to add additional reference types, and to place references in line.

The principal 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).

9.1.1 Defining bibliographies

Bibliographic styles can be declared in Mint using the statement `NewBib`. This takes several parameters that control the way in which citations will appear, and the way in which the entries will be sorted. For example:

```
@NewBib (SolChem,  
         @""@begin(r)@begin(+)(", @Char (',), @""")@end(+)@end(r)",  
         CiteOrder, Numeric, **)
```

specifies the appearance of the bibliography for the Journal of Solution Chemistry. The style is named `SolChem`, and the appearance of citations is determined by the second, third and fourth parameters. The second parameter specifies the string that will appear before the citation, and the fourth specifies the string

that will appear after the citation. The third parameter is used to separate citations if several appear in a list. Thus the citations appear as raised values in parentheses, like this^(1,2). The appearance of the citations values is determined by the sixth parameter — **numeric** specifies roman numbers, **alphabetic** specifies the author name followed by the last two digits of the year, and **shortalphabetic** specifies the first three letters of the author name followed by two digits of the year. <<I really should have a conversion there instead.>> The fifth parameter specifies the way in which the entries are ordered: **codeword** causes them to be sorted by code word, **citeorder** by citation order, and **keyword** by keyword.

9.1.2 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 **NewBibCollection** statement:

```
@NewBibCollection (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.

9.1.3 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)
```

Up to 16 citations can be included in these statements.

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.

9.1.4 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)
```

If you do not pass a parameter to the `bibinclude` statement (that is, you write `@bibinclude` or `@bibinclude()`) then Mint will assume you want the `standard` collection.)

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

```
@Make (Report, @""BibStyle=So1Chem")
```

The style can be set before the bibliography is defined. Initially default values are assumed for the parameters of the bibliography, and then they are reset when the `newbib` statement is made.

9.1.5 The standard bibliographies

The following bibliographies are defined:

```
@NewBib (StdNumeric,  
          @""@begin(r)[", @Char (',)@Char (Sp), @""]@end(r)",  
          KeyWord, Numeric, **)  
@NewBib (StdAlphabetic,  
          @""@begin(r)[", @Char (',)@Char (Sp), @""]@end(r)",  
          KeyWord, Alphabetic, *****)  
@NewBib (CACM,  
          @""@begin(r)[", @Char (',)@Char (Sp), @""]@end(r)",  
          KeyWord, Numeric, **)  
@NewBib (IEEE,  
          @""@begin(r)@begin(+)", @Char (',), @""@end(+">@end(r)",  
          CiteOrder, Numeric, **)
```

Two collections of macro definitions have been written, for the `StdAlphabetic` style and for the `StdNumeric` style. If you want to create a bibliography, you should include the appropriate library, which is `StdAlphabeticRefs` or `StdNumericRefs`:

```
@Library (StdNumericRefs)
```

or somewhat better

```
@Library (@Value(BibStyle)Refs)
```

9.2 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 is 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.

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

9.2.1 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.

9.2.2 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 advice 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.)

9.2.3 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)
```

If **indexinclude** is not passed any parameters (that is, you write **@indexinclude** or **@indexinclude()**) then Mint will create the index for the **standard** collection).

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 Mint automatically 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=IX00008,
          seealso=Uncle Ben, notes=Green and firm)
```

(The label identifier is created by Mint; users should avoid creating labels comprising the letters IX followed by five 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.

(I expect eventually to have `style2`, `style3`, etc., each providing different forms of index.)

The default index style is `style1`.

9.2.3.1 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. Fine details of the layout of the index are left to the user, because a number of macro calls are output, which Mint expects will be supplied.

The following example shows what Mint generates. Assume that the manuscript is:

```
Mint is fairly lax about the way that it uses the word @i(default).
It is used as an environment@index(Default, environment)
and also as a layout@index(Default, layout). When it is used as
an environment@index(Default, environment, style=boldpageno) it
```

creates run-of-the-mill paragraphs, but when it is used as a layout@index(Default, layout) it provides fairly flexible page layouts. Other environments are the @t(describe) environment@index(Describe, environment) and the @t(example) environment@index(Example, environment).

The output created by index include is

```
@init()  
@beginentry(D)  
Default@\environment @pageno(IX00001), @boldpageno(IX00003); layout @pageno(IX00004)  
Describe@\environment @pageno(IX00002), @pageno(IX00005)  
@endentry()  
  
@beginentry(E)  
Example@\environment @pageno(IX00006)  
@endentry()
```

The labels will have been attached to the appropriate parts of the document.

The style parameter determines how the reference to the entry will appear in the index. Mint generates a call to a macro having the same name as the actual parameter, and expects that the macro will convert the label in an appropriate way. For example, if the style parameter is ref the reference will appear as a section number. You can provide your own conversion macros if you wish. You can obtain bold page number entries by defining

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

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

The output is intended to be formatted by the verse environment. For example, if you declare

```
@form(beginentry,1,@""@begin(verse, facecode t, width 5in, tabclear,  
tabset 1.5in, extraleftmargin 2.5in)")  
@form(endentry,,@""@end(verse)")
```

your output will come out like

Default	environment 121, 121; layout 121
Describe	environment 121, 121
Example	environment 121

Eventually I will supply a library of macros to help create indexes.

9.3 Tables of contents

Tables of contents are generated by Mint in two stages. First, if some tables of contents have been declared, Mint scans the presentation when it is complete and collects up information from it; then Mint outputs the information it has gathered in the form of macro calls. The user is expected to provide definitions of these macros that then cause the table of contents to be laid out appropriately. (Of course there is a library that hides most of these details away.) The following document types create tables of contents: report, thesis and manual.

This section first of all describes how to declare tables of contents, and then how the macro calls are generated.

9.3.1 Declaring tables of contents

A table of contents is declared by the statement **NewContentsTable**, which takes two parameters. The first is an identifier for the table, and the second specifies whether the search for entries for the table will look for captions or not. More details are given later about how this search is done. For example, to declare a table of contents named **sections**, do

```
@NewContentsTable (Sections, False)
```

If the second parameter is **false**, it may be omitted.

Once the table has been declared, a collection of environments is associated with it, for example

```
@AssocContents (Sections, Chapter)
@AssocContents (Sections, Section)
@AssocContents (Sections, SubSection)
@AssocContents (Sections, Paragraph)
```

During the scan of the presentation, Mint acts as follows. If the second parameter of the **NewContentsTable** is **false**, the information for the table of contents is extracted directly from text of the specified environments; if it is **true** then the text is extracted from the **caption** environment that is assumed to be nested within the specified environment.

Mint does not perform a simple textual extraction; instead it scans the text and ensures that only text which physically appears in the environment is extracted. Thus footnotes in the text are ignored.

9.3.2 Generating the table of contents

Once the information has been gathered up, Mint then generates a sequence of macro calls. The first call is on a parameterless macro `TCInit`. Then, for each table of contents in the order in which they have been declared, Mint will generate the following calls.

1. A call on a parameterless macro. The identifier of the macro is of the form `BeginXXX`, where `XXX` is the name of the table of contents;
2. A sequence of calls on macros with two parameters. The identifiers of the macros are of the form `TCYYY`, where `YYY` is the name of the environment that is contributing text for the next entry in the table of contents. The parameters are as follows: first, a label that Mint automatically attaches to the box in the presentation; and second, the contents of the box;
3. A call on a parameterless macro. The identifier of the macro is of the form `EndXXX`, where `XXX` is the name of the table of contents.

For example, the sequence of macro calls generated for the first part of this manual were

```
@tcinit()  
@beginsections()  
@tcchapter(TC00001, @""Notes about this implementation")  
@tcsection(TC00002, @""Changes since the last release")  
@tcssubsection(TC00003, @""Syntactic changes")  
@tcssubsection(TC00004, @""Galley properties")  
@tcssubsection(TC00005, @""Units of measurement")  
@tcssubsection(TC00006, @""Fonts")  
@tcssubsection(TC00007, @""Tables of contents")  
@tcssubsection(TC00008, @""State files")  
@tcssubsection(TC00009, @""Separate formatting")  
@tcsection(TC00010, @""Quirks and Oddities")  
@endsections()
```

The library `ContentsMacros` contains the collection of macro definitions that were used for this document. Its contents are

```
@form(tcinit,,  
      @""@begin(pageheading)  
        @begin(align, tabdivide 1)@r{@nconv(romanlc,pageno,)}} @end(align)  
        @begin(align, tabdivide 1)@r{@nconv(romanlc,pageno,)}}@\\@end(align)  
      @end(pageheading")  
  
@form(beginsections,,  
      @""@majorheading(Table of Contents)  
        @begin(align, facecode r, width 5.5in, tabdivide 1)")  
@form(beginfigures,,)  
@form(begintables,,)  
  
@form(endsections,,  
      @""@end(align"))  
@form(endfigures,,)
```

```
@form(endtables,,)

@form(tcchapter,
      @""lab,string",
      @""@vsp(0.25in)
      @b(@ref(@value(lab)) @value(string) @pageno(@value(lab)))@\")
@form(tcsection,
      @""lab,string",
      @""@hsp(0.25in)@ref(@value(lab)) @value(string)
@pageno(@value(lab))@\")
@form(tcssubsection,
      @""lab,string",
      @""@hsp(0.50in)@ref(@value(lab)) @value(string)
@pageno(@value(lab))@\")
@form(tcpparagraph,
      @""lab,string",
      @""@hsp(0.75in)@ref(@value(lab)) @value(string)
@pageno(@value(lab))@\")

@form(tccaption,
      @""lab,string",)
```

Note that no entries are created for figures or tables. I could have removed the definitions of these tables from the state files, but felt that it was easier done here.

Part Ten

Decorations

This part of the Reference Manual describes how to draw borders round boxes and page areas, and how to produce coloured documents (if you have a device that is able to paint in different colours).

10.1 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 3 on page 42. 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}
```

Similarly every page area has a border between the outside of the area and the inside; slugs and boxes are placed within the inside border. The size of the border is specified by the **addareainfo** statement described in section 8.2.2, and normally is zero.

The border style of a box or a page area specifies the appearance of the border; it is possible to construct different border styles. The border is drawn on the inside of the box, and 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 for a box 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}
```

The border style for a page area is specified by the **addareainfo** statement.

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

10.1.1 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.

10.1.1.1 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. |

10.1.1.2 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. (Colours are described in the next section.) 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 8/100th inch of solid black, 2/100th inch that are blank, and a 2/100th inch of dashed line encased by two 1/100th inch black lines. The pattern specifies the appearance of a pattern, from the outside towards the inside of the box.

10.1.1.3 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.
```

10.2 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 provide a means of overlaying objects of one colour by objects of another colour. Objects are either completely transparent and colourless, 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.

10.2.1 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 is 0 for red, 40 for yellow, 80 for green, 120 for cyan, 160 for blue, and 200 for magenta; intermediate values represent mixtures of hues. The colour's saturation determines the contribution of the hue to the colour, with 0 contributing no hue, and 255 being fully

saturated. The colour's brightness is specified by a number between 0 and 255. In terms of these parameters (in the order **hue**, **saturation** and **brightness**) white has values (0,0,255); black has values (0,0,0); full saturated red has the values (0,255,255); full saturated green has the values (80,255,255); and full saturated blue has the values (160,255,255).

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

```
@NewColour (BrightGreen, 80, 255, 255)
```

This specifies that the identifier **BrightGreen** 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).

10.2.2 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.

10.2.2.1 Associating colours with page areas

A colour can be associated with each page area. **Transparent** is associated by default; the colour is changed if a layout routine is passed an area parameters object that specifies a colour. Section 8.2.2 gives more details.

10.2.2.2 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.

10.2.2.3 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.

10.2.2.4 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

```
@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.)

10.2.3 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 8.2.2.2; 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 are laid down in the case where one image overlays another.

Part Eleven

Devices

Mint allows output to be generated for several different devices, simultaneously if needed, and allows a presentation created for one device to be cross proofed onto another device. Devices are collected into classes, such that all the devices in a class have the same driver and the same font format; they differ from each other in secondary characteristics, such as the size of paper or screen they use. This section first of all shows how devices can be declared, and then how output intended for one device can be transformed so that it is suitable for another device.

11.1 Device definitions

Mint has built into it an understanding of how to generate output that can drive certain classes of device, and an understanding of how to interpret font information in different representations. Adding more drivers and representations requires programming effort, but Mint does allow some flexibility by allowing you to specify the properties of *device classes*. Particular devices are declared to belong to some class, and have certain other explicit characteristics. In this section I first show how device classes are declared, and then how devices are declared.

11.1.0.1 Device drivers

Mint has the following drivers:

PressFile	This driver is able to convert the internal representation of a presentation into a Press File. Not all the information in the representation can be converted; in particular the colour <code>greyht</code> is handled as a mosaic of half tone grey patterns. Colour information is placed in the press file, though the Dover drivers ignore it.
PerqScreen	This driver is able to convert Mint's internal representation into a form suitable for display on the Perq screen. The final stage of this driver is an operating system specific interface, which is not strictly a part of Mint, so the same driver is used for both POS and Sapphire. Colour information is ignored.

11.1.0.2 Font formats

Mint understands the following font formats:

- | | |
|--------------------|---|
| XeroxFormat | This is the format that is present in the <code>fonts.width</code> file. Mint uses the bounding box information and the width information from the file, and supplements it with character information if it is available. This additional information is a superset of the information in the <code>.TFM</code> files, from which information is extracted to allow Mint to operate with Metafont fonts. |
| KstFormat | This is the format in which font information is stored on the Perq. The information describes the bounding box and the widths; unfortunately the format is deficient in several ways, and the baseline information is generally incorrect. This causes characters to be displaced vertically. The extra character information could correct this, but I've not yet felt it worth the effort to accumulate it. |

Since Mint can have all the character information it needs presented to it via statements, I should define a **MintFormat**, which uses no other external source of information. This could be very useful for getting Mint to drive non-standard devices.

11.1.1 Device classes

A device class is characterised by having some device driver, font format, resolution in the *X* and *Y* directions, an *inner scale*, and a specification of the raster operations the device can perform. The inner scale is used to prevent overflow of two 16-bit values that measure the horizontal size of a slug, and its vertical offset from the start of its box. For example, the resolution of the **PressFile** device class that is declared as a part of the normal configuration of Mint is 2540 units to the inch. To allow boxes that contain slugs to grow to be as large as five pages, the inner scale is set to 5. Clearly a better solution would be to use 32-bit quantities — a trivial change, though tedious to effect.

Examples of device class declarations are

```
@NewDClass (PerqScreen, PerqDriver, KstFormat, 90, 96, 1, #377)
@NewDClass (PressFile, PressDriver, XeroxFormat, 2540, 2540, 5, #20)
```

The resolutions in the *X* and *Y* directions are specified in terms of some unit of measure; the only requirement that Mint places on this is that all other units are expressed in terms of it. In the case of the standard configuration, inches have been chosen, so the values above are the number of rasters per inch.

11.1.2 Devices

Devices may be declared quite freely in Mint. There are some restrictions on altering device properties once galleys have been created that are associated with these devices, however. The **newdevice** statement declares a new device:

```
@NewDevice (Perq, PerqScreen, 8.4 in, 9.5 in)
@NewDevice (Dover, PressFile, 8.5 in, 11 in)
```

The first parameter is the identifier for the device, the second is the device class to which it belongs, and the third and fourth parameters are the values of **pagewidth** and **pageheight**. Normally the page area sizes are expressed in terms of these quantities, though there is nothing intrinsic in these values.

If the **newdevice** statement identifier refers to a previously existing device, then Mint will alter its properties. If, however, galleys have been created that use the device, the attempt to change the properties will fail if any slugs have been created for the galley. For this reason, if not for others, all **newdevice** statements should be at the beginning of the manuscript, or in the definitions file.

(As it happens, Mint will create new devices automatically under Sapphire if the window changes size.)

11.2 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 **P1ot** and **DP** environments, the scaling can be done with reasonable accuracy; however, Mint is not able to scale fonts, and for this reason it 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 (PerqScreen, Dover, TimesRoman10)
```

specifies that when some device that belongs to the `PerqScreen` class has been specified as target device, and the document is viewed on the Dover, then `TimesRoman10` will be used as the default font. A specific replacement can also be made:

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

specifies that if some character for a `PerqScreen` target device, is in `Gacha9`, then it is to appear as `Gacha8` on the Dover.

One can build up a reasonable set of cross proofing font pairs — one such collection is in the library `CrossProofTR`, which improves the appearance of documents prepared for the Dover using the default Times Roman fonts when they are viewed on the Perq.

Note that the viewing device is specified as an explicit device, and not as a member of some device class. Because of this a situation can arise in which cross proofing fonts have not been specified for the viewing device (in particular this will occur if the screen size is changed under Sapphire, since Mint automatically creates a new device for each change). In this case Mint will use the cross proofing fonts for some arbitrary device that belongs to the same class as that of the viewing device.

There are some subtleties in cross proofing documents when character substitutions have been made. Consider the following example. A document is prepared for the Dover using the Times Roman fonts, and ligatures have been specified. In particular, the sequence `f` followed by `i` is to be replaced by `fi`, which in the Times Roman fonts has the ASCII value #24. In the internal representation of the document are occurrences of this character. If the document is to be viewed on the Perq, using the Metafont `Cmr` fonts, we want the `fi` ligature to appear. Unfortunately the ligature is at ASCII value #174 in these fonts, so it is necessary to have a `substitutefchar` in the viewing fonts as well. Mint applies two series of transformations: the first using the substitutions for the target device, and then a second set using the substitutions for the viewing device. (In fact all the standard character manipulations of kerning, substitution, etc, are performed on both sides of the transformation.)

Part Twelve

Mathematical Typesetting

Mathematical typesetting is described in this section of the manual. Mathematical typesetting uses two environments — a slug environment for in-line formulae, and a box environment for displayed formulae. In these environments Mint applies special scanning rules, that mean that formulae are laid out according to standard typesetting rules without you needing to be aware of these rules. The first section describes straightforward typesetting, which should be sufficient for most needs. The next section describes typesetting rules that are needed for more sophisticated formulae. The last section explains in more detail the underlying rules that Mint employs.

<<Extensive revision of this section of the Manual will be happening soon. Also, several parameters still need to be tuned — especially those for accents. What is here now is mainly truthful.>>

12.1 Mathematical Typesetting

Mint provides a range of mathematical typesetting facilities which are similar to those provided by T_EX — 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 T_EX, 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 T_EX, so that (potentially, at least) any formula that can be typeset using T_EX 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.

12.1.1 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

$$\binom{p}{2} x^2 y^{p-2} - \frac{1}{1-x} \frac{1}{1-x^2} \quad (12-1)$$

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 12-1 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 superscript 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 \TeX the collection of rules that need to be applied is determined by the *style* of the formula; in-line formulae are in *text style* (`StyleT`), and out-of-line formulae are in *display style* (`StyleD`). Several other styles occur in formulae — *script style* (`StyleS`) uses a smaller font size for superscripts, and *script script style* (`StyleSS`) 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 environments¹³. The different classes of symbol are **ordinary** (corresponding to variables), **operators** (such as \sum), **binary operators** (such as $+$ and $-$), **relational operators** (such as $=$ and \leq), **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 [TEX, Chapter 18].

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

$$L = \begin{pmatrix} 1+x & \frac{(20)}{x+y} & 3.14159 & \frac{1}{a+\frac{1}{a+b}} \\ 3_i & 4+5+6 & \sin x & \sin y \end{pmatrix}$$

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, 1abel 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)}
```

¹³ 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.
```

12.1.2 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(s in x)` 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 `cb break`; this has no effect if the formula does not need to be broken¹⁴.

A formula is displayed out of line using the `maths` box environment¹⁵. 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 that of any other box environment.

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.

¹⁴ Console yourself that it is easy to check your text using cross proofing.

¹⁵ `math` is also allowed.

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

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 subintervals¹⁶.

Figure 6. Example of a mathematical formula in a figure

12.1.3 More complex formulae

12.1.3.1 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 on request¹⁷. For example, to obtain an in-line fraction, you write `@m(@fract(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.

<code>@sup(a,b)</code>	This superscripts the first argument; for example a^b . The amount the superscript is raised is a function of the style of the formula.
<code>@sub(a,b)</code>	This subscripts the first argument; for example a_b . The amount the subscript is lowered is a function of the style of the formula.
<code>@supb(a,b,c)</code>	This both superscripts and subscripts; for example a_c^b . The amount of raising and lowering depends on the style of the formula.

¹⁶ From Ralston, A First Course in Numerical Analysis. McGraw-Hill.

¹⁷ And after paying a suitable fee.

@fract(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 **@fract**, but there is no divisor line, and the formula rests on the reference line (see figure 7).

@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

$$\left(\left(\left(\begin{matrix} a \\ b \end{matrix} \right) \right) \right)$$

the 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)**).

@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 **@mcoll** 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, lim, 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.

12.1.3.2 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) - @fract{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:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad (12-2)$$

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 12-2 on page 141 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.

12.2 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.

12.2.1 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 a private font `maths`. 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 `maths` 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`.

The only difference between the `maths` font and the other fonts is that the `maths` font is a *private* font; that is, it is associated with a font family using `assocprivf` rather than `assocfont`. Apart from this all the fonts used for mathematical typesetting are handled in the same way as any other fonts — they have their font characteristics specified in the same way, and they can have other characters, gaps or icons substituted for characters in them. The slug environments `m0`, `m1` and `m2` can be used in the same way as `r`, `i`, etc. Note that the fonts `m0`, `m1`, `m2` and `maths` need to be associated with each font family that is to be used for mathematical formulae¹⁸.

12.2.1.1 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 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

¹⁸ In particular, you cannot put formulae into footnotes and page headings and footings unless you have associated the fonts with the font families of the appropriate galley.

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

(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

12.2.2 Defining symbols

Mint's spacing rules are applied by classifying each symbol that occurs in a formula, and thereafter by 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	used for relational operators;
binop	used for binary operators such as +;
monop	used for monadic operators (Mint will change a binary operator to a monadic operator according to simple parsing rules);
op	used for operators such as sin;
ord	used for multi-character variables;
ord1	used for single-character variables;
open	used for left parentheses and other left delimiters;
close	used similarly for right delimiters;

num	the style automatically given to numbers;
word	used for components of programs;
punct	used for punctuation.
flush	used for tabulations and for special symbols, when you need to control the amount of space precisely.

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 so regard it as an **ord1** or **ord** (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.

12.2.2.1 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, \mathbf{x} , $\mathbf{\tilde{x}}$, and $\mathbf{x'}$ are all inflected. Mint provides a general means of inflecting symbols, and allows several simultaneous inflections, as in $\mathbf{\tilde{l}'}$; 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 **over**, **under** 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 \bar{o} \bar{o}dot)` produces \hat{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 \uparrow+y \uparrow)` to obtain $\hat{x} + \hat{y}$.

12.2.2.2 Replacement text

Many symbols that occur in mathematical formulae, like E_i , are thought of as single objects even though they are composite. Although you can use a macro for these composite objects, the presence of the @ and the parentheses of the macro call make formulae more difficult to write, and more difficult to read in the manuscript. Mint provides a simple means of introducing new symbols that expand into a primitive mathematical statements, without having to use the heavy-weight support of the macrogenerator. The statement `edef` allows you to specify the replacement string for a lexeme, such as

```
@EDef (Ei, @"@sub(E,i)")
```

After this statement you can write `@m(E i=0)` and you will obtain $E_i = 0$ (note the need to quote the second argument to avoid it being interpreted within the `edef`). The replacement will only occur within the mathematical environments, and the symbols are case-sensitive.

It would be good to be able to say that `edefs` could be used without caution, and used in all the ways you would expect them to be used. Alas, that isn't the case. There are some dirty interactions with the macrogenerator that are difficult to describe; however the problems will go away when the macrogenerator goes away. In the meantime you should only use `edefs` for simple substitutions like the one above.

12.2.3 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.

12.2.4 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*, without however 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=stylelet)
```

12.2.5 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(fract, style = stylelet)
1, b+c
@end(fract)
```

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 <code>StyleT</code> , <code>StyleD</code> , <code>StyleS</code> and <code>StyleSS</code> . The specified style is imposed on the formula type. See below for easier ways of using this style parameter.
Xposn	This takes values from <code>Left</code> , <code>Centre</code> ¹⁹ and <code>Right</code> . It specifies how the mathematical formula will be positioned inside the parent formula: <code>Left</code> flushes the formula to the left; <code>Right</code> flushes it to the right; and <code>Centre</code>

¹⁹ also `Center`.

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 12.1.1 had **Xgap** and **Ygap** specified to be 0.5 quad.

Type This takes one of the values **rel op**, **binop**, **op**, etc., (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 **rel op**; then you write

```
A @begin(col, type=rel op) 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

$$A \xrightarrow{} B$$

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)**, for values of **s**, **t**, **x** and **y** occur frequently, Mint provides abbreviations. In each of these cases you can write, for example, **@style(formula)**, **@ord(formula)**, **@left(formula)**, and **@below(formula)**. In all cases the identifier you use is the same as the **style**, **type**, **xposn** or **yposn** you require; alas, since **centre** is both an **xposn** and **yposn** parameter, you must write **@xcentre(formula)** or **@ycentre(formula)** (or **xcenter**, **ycenter**).

You can modify any of the maths environment parameters, using **@begin(parameters)**, in the usual way. (However, if you use write, for example, **@begin(stylet, style=styled)**, you will get **styled**.)

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 `@fract`, 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(fract, style styled)
    @left(1), @sub(a,2) +
      @begin(fract, style styled)
        @left(1), @sub(a,3) +
          @begin(fract, style styled)
            1,b
          @end(fract)
        @end(fract)
      @end(fract)
}
}
```

(I never said it was going to be elegant) produces

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

12.2.6 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, `@\v` 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@\v+d+e}
```

produces two formulae, and

```
@maths{@fract(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

$$E = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \quad \text{and} \quad F = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad (12-3)$$

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
123@>456@< 123@>456		123456	
123@>456@< 123@>456@\ 123@>456@<		123456	
	123	456	

Input			
@>123@<456		123456	
@>123@<456@<		123456	
@>123@<456@\ @>123@>456	123	456	
@>123@>456@< @>123@>456@<	123	456	
@>123@>456@< @>123@>456@<\	123		456
@>123@>456@< @>123@>456@<\		123456	
@>123@>456@< @>123@>456@<\		123456	
	123	456	

Input			
@\123@<456		123456	
@\123@<456@<		123456	
@\123@<456@\ @\123@>456	123	456	
@\123@>456@< @\123@>456@<	123	456	
@\123@>456@<\	123		456
@\123@>456@<\		123456	
@\123@>456@<\		123456	
@\123@>456@<\	123	456	

12.2.7 Equation counters

The counter associated with equations is `EquationNo`; it has no parent counter in document types without sections or chapters (for example `text` and `slides`) — in such documents it counts sequentially. In document types with sections or chapters its parent counter is `SectionNo` or `ChapterNo`, respectively; thus in a thesis, for example, it is reset to 1 at the start of every chapter. Mint also defines a pseudo-counter, `Equation`, which yields a reference comprising the chapter (or section) number, followed by the equation number. This is used for numbering equations in the `maths` environment, and is also used by the `eqn` macro.

The best way of understanding these statements is by example. The macro `eqn` is defined as

```
@form (eqn, lab, @""@nconv (equationstyle, equation, @value (lab)))
```

and if we define a macro `bareeqn` as

```
@form (bareeqn, lab, @""@nconv (equationstyle, equationno, @value (lab)))
```

then we will get for

```
@Eqn (TEX-p68) is the full reference, and @BareEqn (TEX-p68) is the number  
of the equation within the chapter
```

the statement that 12-1 is the full reference, and 1 is the number of the equation within the chapter.

(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.)

12.3 Really advanced features

The features that have been described above should allow most users 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.

12.3.1 Non-fanatics stop here

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.

12.3.2 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. The diagram below shows the ($XRe1, YRe1$) vectors for a simple formula. The dimensions have been distorted to show the origins and bounding boxes.

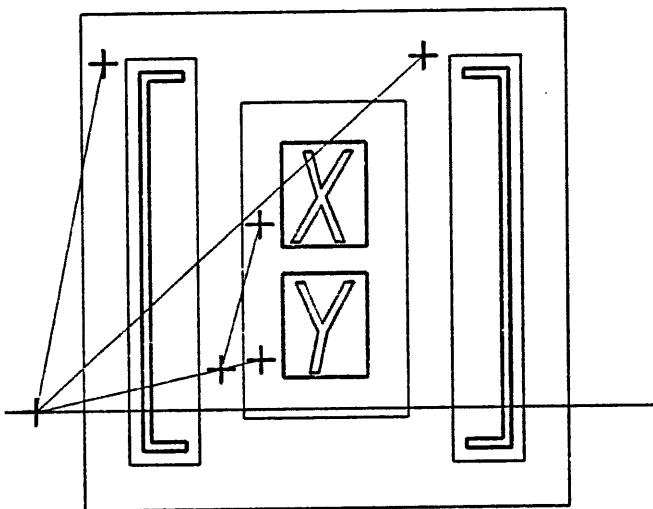


Figure 7. Mathematical vectors for $\begin{bmatrix} X \\ Y \end{bmatrix}$.

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 in which 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 12.2.5 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.

12.3.3 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.

12.3.4 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 several new types. These are as follows: **Num**, the type of a number, which allows a different space to be placed between the 2 and the *i* in $2i$ than that between the *i* and *j* in ij (note how this also differs from ij); **Ord1**, which differs from **ord** because all those computer scientists want to have multi-character identifiers, such as **wirein**, and need wider spaces than in **wi**; 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**.

12.3.5 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.

12.3.5.1 Spacings

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 $A = B$.

12.3.5.2 Rows and columns

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 12.2.5. Note that the parameter to `hsp` and `vsp` can be negative.

12.3.5.3 Positioning parameters

The `xposn` and `yposn` environment parameters take, in addition to the values specified in section 12.2.5, explicit values and *label values*. An explicit value is given as a value of some length, for example `3inches` or `4ems`. These specify the distance the object will be placed from the origin of the parent's 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`.

12.3.6 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 T_EX 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.
- P1** The extra height of a superscript in DMode above the position it would have with style parameter YPosn equal to Below.
- 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.

Part Thirteen

Alternative interpreters

Mint provides a simple interface to interpreters. It expects that these interpreters will construct objects that will later come to be displayed in presentations. The interpreters should, therefore, understand the requirements of the device drivers and font formats, and they should respect the stipulations that Mint makes about the size and position of the objects they display. Mint supplies interpreters with a variety of services for scanning text, managing memory and creating output for devices, but there are no requirements that the interpreters use these facilities, and they may, for example, use their own syntax or their own channels to the devices. In this section I describe two interpreters that have been interfaced to Mint.

13.1 Line drawings

Both the interpreters that have been interfaced so far are for line drawings. They illustrate a range of interfacing strategies: DP has a very simple interface, and both Mint and DP inhabit different universes, with little communication between them. Nonetheless the interface is effective and efficient. The Plot interface is more extensively programmed, because there was no Plot driver for the Perq. It was therefore worth while building an interface that incorporated the device drivers.

13.1.1 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 which is finally needed.»

Part Fourteen

State Files and Libraries

This part of the reference manual describes a collection of facilities that are useful for managing document designs. A document design is implemented by a sequence of Mint statements that describe the appearance of environments and pages, and describe the way that a document is built from these environments and pages. In most cases users of Mint will use several standard designs that are distributed as a part of Mint, possibly extended with additional definitions. In section 14.1 I describe the facilities that can be used to maintain these definitions, and in section 14.2 I describe some of the standard collections of definitions. Section 14.3 contains a detailed description of the structure of the files which are used to make the standard documents. It is unlikely that this section will be needed by most users, though it forms essential reading for system maintainers, and will yield lots of insights for the curious.

14.1 Definition files

Before processing a manuscript file, Mint reads a *state file*; this is a binary file that sets up Mint's internal state. State files are useful because they can establish Mint's internal state far more quickly than if Mint had to interpret the corresponding statements, and because they can be used for encapsulating definitions that are used to describe document designs. With the appropriate supporting tools, state files can complement manuscript files as components for creating documents.

In this section I describe the basic features of state files, indicate how the state files for the standard document types are created, and describe the tools that Mint supplies to help tailor state files by adding additional definitions.

14.1.1 Standard state files

For each different document type, in each of its different forms, and for each different device, there is a different state file that contains, in binary form, the definitions that specify the properties of the document. Thus the state file for creating a manual, form 0 for the Perq differs from the state file for creating a manual,

form 0 for the Dover. Mint determines which state file to use by examining the `make` statement; the first three parameters are concatenated together (using defaults, or interacting with the user if necessary), to create the state file name. For example, if the `make` statement is

```
@Make (Manual, 0, Dover)
```

then Mint will read the file `Manual0Dover.State`; if it cannot find the file then it is unable to proceed. There is nothing sacred about these three parameters except for the way the defaults are supplied: if no second parameter is given, then 0 is assumed; if no third parameter is given, then if the device request on start-up was a specific device (e.g. P, D, or `IGE11atio`, for example) that is the third parameter, otherwise (X was given, and) the third parameter is `perq`. Should a third parameter be given, then the previous sentence applies, except that “`perq`” is replaced by the specific third parameter. If there is no `make` statement, then Mint provides a default of `@make(text, 0, perq)`. (If all the parameters are empty, then Mint doesn't read a state file. This is used to boot-strap the standard state files; see below.)

If there are additional parameters in the `make` statement, they are packaged up and handed on to the `begin` statement for the document. The way to understand what happens is to realize that the `make` statement is a macro with four parameters; its effective definition is

```
@Form {Make, @"(DocumentType, Form=0, Device=Perq, Rest), @"(Some Body)}
```

and the `rest` parameter is handed on. To ensure that everything works in the right way, you should do something like

```
@Make (Slides, Rest=@"{ImageColour=Red, Width=5 inches})
```

14.1.2 Creating standard state files

Under normal circumstances the state files you need will have come along as part of the standard release of Mint. If it complains that it cannot find the state file, though, you will have to construct one. In this section I describe how to do this task starting from several ASCII files I supply; if you don't have these you have no chance whatsoever of getting yourself off the ground. Read section 14.3 if you don't believe me.

The files to create the state files are called *state creation files*; they all have the extension `.Mint`. There are state creation files for all the document types that are supported by Mint at present. Altogether there are some 120 files you need, and there are many levels of `include`. However provided you have all the files, all you need to do to create the state file for (say) a manual, form 1, for the Dover, is to start up Mint, and when it requests the file name, give it `Manual1Dover`, and device D. It will then go about its business for some minutes, and eventually write off a state file with the name `Manual1Dover.State`. You can then exit from Mint (it does create a document, but it consists of just one single fullstop).

The state file format is fairly dependent on the version of Mint you are using, so when a new version of Mint comes along, you should retrieve new versions of the state files, or go about creating new ones yourself. If Mint finds that its own version number differs from the version of Mint that created the state file, it will issue a warning. It may later issue a fatal error, or even collapse, if there are debilitating incompatibilities.

14.1.3 Low-level state file manipulations

There are two low-level statements for manipulating state files. Normally you will use the better structured facilities described in the next section, so this is for the hackers.

The `make` statement can be replaced by the `readstate` statement; this takes a single parameter which is the name of the state file to read. Thus `@make(text,0,perq)` is equivalent to `@readstate(text0perq)`. Like the `make` statement, this should be the first statement in your manuscript file.

Mint's internal state is a merge of the state read from the state file and state created as a result of executing statements, such as `form`, `newcolour`, etc. (Read section 14.3 to see all the statements that can contribute to the state.) At any time you can dump out the current internal state using the `dumpstate` statement. The standard state creation files call `dumpstate` when they have incorporated all the definitions needed for some particular document type into Mint's internal state. A state file is just a snapshot of Mint's internal state which can be reconstructed using `readstate`. Because some parts of Mint's state take a long time to create from statements, you gain a lot by saving the state in binary form (this is especially the case with the font operations, which have been optimized for fast execution once the internal state exists, at the expense of very slow creation of the internal state).

One way of using the state file facility of Mint is to start off with a standard state file, read it in (using `make` or `readstate`), add to it the collection of definitions that you regard as important, and dump it out. You then have a state file that contains a collection of document design definitions. However, if you have any taste, you won't hack it at this level, you will use the facilities described in the next section.

14.1.4 Using definitions files

Assume that you have a collection of Mint definitions that define the appearance of galleys, pages, environments, etc. You should separate these definitions from the body of the manuscript file that uses them for several reasons: you may want to use the definitions for several different documents, or you may want to impose a global style on several different authors, without them having to be aware of the details of the definitions you have used. Mint provides a facility to help achieve these goals. It is based on automating

the reading and dumping of state files; it relies on a good separation of the definitions from the manuscripts that use the definitions.

Let's assume that you have indeed separated out a collection of definitions from the body of the manuscripts that will want to use the definitions. You now need to be able to create the state file automatically, and ensure that if you alter the definitions file, a new state file will get created that contains the changes in the definitions file (very strictly, Mint regards a state file as an optimization of a collection of ASCII definitions).

You achieve this as follows: Put the definitions, preceded by the `make` statement for the document²⁰, in a file whose extension is `.defs`. For example, assume that `Ninerva.Defs` contains a collection of definitions. In the manuscript file that needs `Ninerva.Defs` include the statement `@ReadDefs(Ninerva)`. This statement does the following. Mint checks to see if there is a file `Ninerva.State`. If there is not, it reads the file `Ninerva.Defs`, and at the end it dumps out its current state into file `Ninerva.State`. Mint then continues processing the manuscript. If there is a file `Ninerva.State`, Mint checks to see if the current version number of `Ninerva.Defs` and of all the files that it includes match the version numbers stored in the state file; if they do then Mint uses the state file, otherwise it renames the state file (to `Ninerva.State$`) and processes `Ninerva.Defs` as though the state file did not exist.

In this way Mint supplies you with mechanisms to separate out the design abstractions from the documents that use them, and allows you to have some confidence that the optimizations that Mint performs in order to keep the definitions in binary form will continue to reflect the current state of the definitions files.

For an example of these facilities, read the file `RefMan.Defs`, which contains the definitions for this manual.

14.2 Libraries

Mint library files provide a number of facilities to help create documents. Library files consist of Mint statements that are interpreted in the usual way; normally they comprise some statements that set up Mint's internal state, or they provide collections of useful macros. They can be incorporated into manuscript files by using the `Library` statement, for example

```
@library(SpiceTitlePage)
```

²⁰ Actually it can start with another `readdefs` statement. I'm sure that there will be plenty of people out there making sure that the statements I now make really do apply to nested `defs` files.

All library files have the extension .lib; thus the statement incorporates the file SpiceTitlePage.Lib. The library is incorporated into the document as though the appropriate `include` statement had been made; however, Mint searches to see if the library has already been incorporated, and it only includes the .library if it is not there.

The libraries that are available now are

Cmr10Kern	This library contains the many statements that define the kerns and ligatures for the Cmr fonts. You should include this library if you are using the <code>\TeX Cmr</code> fonts. This library, together with the TimesRomanKern library described below, has been produced mechanically from the .TFM files that describe the <code>\TeX</code> fonts, so other libraries can be created quite easily.
ContentsMacros	This defines a set of macros that set out the tables of contents. The table of contents for this manual has been produced using them; they provide entries for sectioned statements, but not for tables or figures; however, you should have no problem adding further macros if you need them. Section 9.3 contains a description of the way tables of contents are produced.
CrossProofTR	If you are creating a document for the Dover, using the Times Roman fonts that Mint supplies as the defaults, and you wish to see what the document looks like on the Perq screen, then you should use this library. It sets up several crossproofing statements for the <code>\TeX Cmr</code> fonts, which almost acceptably match the Dover Times Roman fonts, and modifies them so that most of the incompatibilities between ASCII encoding and <code>\TeX</code> encoding are handled. The mapping isn't complete, though, so I need a public-spirited person to improve them. Also, the maths extension font for the Perq is the wrong size, which causes mathematical formulae to appear strange.
CrossProofSlides	This plays a similar part to <code>CrossProofTR</code> , for the <code>slides</code> document types, using the default fonts supplied by Mint.
SpaceFont	This provides a single macro, <code>SetSpacing</code> , that modifies a font so that the gap between individual letters differs from the design gap. The parameters of the macro are a font values object, a device class, a font identifier, and then two lengths, which are the distance that the bounding box of every character will be displaced in the <i>X</i> direction, and the extra translation in the <i>X</i> direction that is used between characters. The final parameter is an integer value between 1 and 7, which should be different for each use of <code>SetSpacing</code> (that's not quite true, but it's a good guide). This parameter allows the same fonts value object to be used with different displacements and translations. An example is useful. Assume that you wish to increase the spread between the characters of the <code>TimesRoman18b</code> font, to lighten the appearance of a title. The following shows what the original unspread font looks like, and what a font that is derived from it looks like:

```

@library (spacefont)
@copyfont (pressfile, titlefont, timesroman18b)
@assocfont (fonts0, 11, f8, timesroman18b)
@assocfont (fonts0, 11, f9, titlefont)

@setfontinfo (pressfile, titlefont, bold, regular,
               18 point, 0, standardr, , standardbbi)
@setspacing (standardr, pressfile, titlefont,
              0.02 em, 0.04 em, 1)

@begin(majorheading, facecode f8)
CARNEGIE-MELLON UNIVERSITY
@end(majorheading)

@begin(majorheading, facecode f9)
CARNEGIE-MELLON UNIVERSITY
@end(majorheading)

```

CARNEGIE-MELLON UNIVERSITY

CARNEGIE-MELLON UNIVERSITY

Good typographic practice only modifies the gaps between letters in three circumstances: the case above, where a heading is otherwise too dense; a small adjustment (usually a reduction) of character gaps throughout the whole of a document; and the use of wider spacing for emphasis, though this practice is almost entirely confined to German-speaking countries. Use this library sparingly.

SpiceTitlePage

This library contains the macros for laying out the Spice title page. It was used for this manual. The parameters it uses are as follows:

title	The title of the document.
author	The author of the document.
dt	The date of the document; if omitted, the current date will be used.
abs	The abstract; it may be omitted.
doc	The document number.
cats	The filing categories; it may be omitted.
file	The location of the machine-readable form; it may be omitted, when the current file will be used.
ext	Nonblank if the document is to get general distribution; otherwise the Spice disclaimer notice will be incorporated.
excl	If nonblank the copyright notice will refer to the author; otherwise to CMU.

This library incorporates three others automatically: `spacefont`, `spicedisc` and `rescrnote`.

StdAlphabeticRefs	This library contains the bibliographic reference macros for the standard alphabetic reference format, and it should be incorporated if you want to cite references. Since it also contains the definitions of which macros are <i>delayed</i> (see section 6.1.3), it should be incorporated before you include your bibliography data base ²¹ . It incorporates another library automatically: <code>standardrefs</code> .
StdNumericRefs	This library is similar to <code>StdAlphabeticRefs</code> , except that it contains the macros for the standard numeric reference format. It also incorporates the <code>StandardRefs</code> library. (Other <code>Refs</code> libraries will become available some day.)
TimesRomanKern	This library contains the kerning and ligature statements for the TimesRoman fonts used in the standard Mint font families. The Xerox TimesRoman fonts have been well hand-tuned, and require only minor second-order modifications of character spacings to make the output pleasing. Rather more important are several ligatures; these are <code>f i</code> which produces <code>fi</code> , <code>ff</code> which produces <code>ff</code> , <code>f l</code> which produces <code>fl</code> , <code>ff i</code> which produces <code>ffi</code> , <code>ff l</code> which produces <code>ffl</code> , <code>--</code> which produces <code>-</code> , and <code>---</code> which produces <code>—</code> . I don't regard these as exhaustive, so you may wish to add your own.
UserFaceCodes	This library contains definitions of the face codes <code>f0</code> ..., <code>f9</code> . It should be included in any document that is using the default user face codes that were imposed by the previous version of Mint.

14.3 Defining state

This section describes the Mint statements that are most likely to occur in state files. The description is tied closely to the standard state files, and in particular to `Manual10Dover.State`. A word of warning: several of these statements do not perform exhaustive checking of their parameters, and in some cases the order of statements is important, so you should check all uses of them carefully before releasing them on unsuspecting users.

14.3.1 Basic definitions

Several definitions are built into Mint, for example some lexemes such as `begin` and `end`, and some statement identifiers such as `include` and `make`. Eventually all lexemes will be read in from the state files, but at present this restriction means that there are some parts of the Mint input language that are not

²¹ A convenient way of incorporating the references libraries is to use `@Library(@Value(BibStyle)Refs)`.

amenable to translation into other natural languages. This is not the case, however, with the in-built statements, since these can be equated to other identifiers using the macrogenerator. For example, if you do not want **include** you can provide a definition like

```
@form( Inclusione,X,@"{@Include(@Value(X))})
```

The statement identifiers that are built in are those that are useful for bootstrapping the rest of the state file; they are: `include`, `readdefs`, `comment`, `make`, `readstate`, `dumpstate` and `specialform`. All others are defined using `specialform`, as described below.

14.3.1.1 The Special Form statement

Each statement that Mint can execute is identified by a small integer number, which is used internally as an index for a `case` statement. The `SpecialForm` statement binds the identifier for the statement, the identifiers for the formal parameters, and a flag indicating whether the statement allows a variable number of parameters, to the `case` statement index. The following is a short extract from the file `BasicDefs.Mint`.

```
@SpecialForm (form, @"(id,params,body), , 17)
@SpecialForm (value, id, , 18)
@SpecialForm (defer, macro, , 19)
@SpecialForm (reinvoke, dm, , 20)
@SpecialForm (newbibcollection, collection, , 21)
@SpecialForm (citeincollection, @"(collection,,,...,,,), t, 22)
@SpecialForm (cite, @"(.,...,.,), t, 23)
@SpecialForm (bibinclude, @"(.,...,.,), t, 24)
@SpecialForm (newcounter, @"(id,within,start=1,change+=1), , 26)
@SpecialForm (next, id, , 27)
@SpecialForm (set, @"(id,value), , 28)
```

If the third parameter is not empty, then the statement can take as many parameters as are indicated, or fewer. **SpecialForm** does not check that the number of parameters is appropriate for the statement being specified.

14.3.1.2 The lexeme tables

The lexeme tables are set up by the `initlexmap` statement (the identifier of which is defined by a `specialform` statement, of course). This statement takes three integer values which specify the module, table and entry to be set by the fourth parameter. The lexemes are not case sensitive. The following is a short extract from the file `BasicDefs.Mint`.

```
@InitLexMap (2, 5, 0, false)
@InitLexMap (2, 5, 1, true)
@InitLexMap (2, 6, 0, disallow)
@InitLexMap (2, 6, 1, allow)
@InitLexMap (2, 6, 2, )
```

```
@InitLexMap (2, 6, 3, true)
@InitLexMap (2, 7, 0, )
@InitLexMap (2, 7, 1, text)
@InitLexMap (2, 7, 2, plot)
@InitLexMap (2, 7, 3, dp)
```

The module, table and entry numbers are built into Mint.

14.3.1.3 Units

Units are declared in terms of some basic unit, which is that used to express the raster unit resolution of the devices. The standard state files use inches for this measure. An extract from the file **BasicDefs.Mint** is:

```
@SetUnit (in, 1.0)
@SetUnit (inch, 1.0)
@SetUnit (ins, 1.0)
@SetUnit (inches, 1.0)
@SetUnit (point, 0.013837)
@SetUnit (points, 0.013837)
@SetUnit (pica, 0.166044)
@SetUnit (picas, 0.166044)
```

14.3.2 Syntax definitions

Statements exist that allow you to create the directed syntax graph used by the parser, and to specify the pseudo-syntactic properties of the environments. These statements expect that other definitions (of counters, for example) have already been made. If you are in doubt about the order that statements should occur in, follow the examples in the standard **.Mint** files.

Since the syntax graph has cycles, it is necessary to introduce the identifiers of the nodes of the syntax graph before the graph is specified; this is done by the **RegisterEnv** statement. The following is an extract from the file **SyntaxDefs.Mint**.

```
@RegisterEnv (default)
@RegisterEnv (align)
@RegisterEnv (itemize)
@RegisterEnv (enumerate)
@RegisterEnv (describe)
@RegisterEnv (maths)
@RegisterEnv (commentary)
@RegisterEnv (textpart)
@RegisterEnv (gloss)
@RegisterEnv (figure)
@RegisterEnv (table)
```

Since Mint performs special actions on some of these environments, the order in which they occur is

significant. The file contains a comment about which environments Mint treats specially. All other environments can be introduced in any order. If you wish to add new notions to the syntax, and **define** and **modify** are not appropriate ways of doing this, you can use **registerenv**, and specify all the properties of the environment. Follow the example of (say) **centre** to see how to do this.

After the environment identifiers have been registered, the syntactic description of the document is built up in several stages, described below.

14.3.2.1 Syntax classes

The syntax is described not in terms of environments directly, but instead in terms of *syntax classes*. These are specified by the **NewSyntaxClass** statement. The following is an extract from the file **SyntaxDefs.Mint**.

```
@NewSyntaxClass (Terminals)
@NewSyntaxClass (Sections)
@NewSyntaxClass (Headings)
@NewSyntaxClass (Items)
```

14.3.2.2 Production rules

Production rules are specified by the **NewProduction** statement. This allows you to create a named production rule, and specify the pseudo-syntactic properties. Examples drawn from the file **ProdDefs.Mint** are:

```
@NewProduction (term1, terminals, , false, false, 0).
@NewProduction (item2, items, flushleft, false, skip, 2, ,
                 terminals, items)
@NewProduction (foot1, foots, default, true, skip, 2, annoteno, crannote,
                 terminals, items)
```

The statement is somewhat arcane, so you are well advised to follow the patterns carefully. The third example above corresponds to the rule

- | 1) | Rule id | Rule | Default Environment | Dominating | Pack | BlankLines |
|----|---------|----------------------------|---------------------|------------|------------|------------|
| | Foot1 | Foots : Terminals Items. | Default | True | Irrelevant | Invalid |
- 2) Automatically increment the counter **AnnoteNo** whenever this rule creates a box.
 - 3) Insert the lexeme sequence **CrAnnote** into the parent environment.

Several of these parameters can be omitted. <<Some changes have been made in this file for the annotation feature.>>

Finally, the environment identifiers are associated with particular production rules (and thereby placed

into particular syntax classes) by the statement `CrSyntax`. The following is taken from the file `TerminalSyntax.Mint`; the other `xxxSyntax.Mint` files are similar.

```
@CrSyntax (centre, term1)
@CrSyntax (flushleft, term1)
@CrSyntax (flushright, term1)
@CrSyntax (plot, term1)
@CrSyntax (dp, term4)
@CrSyntax (verse, term1)
@CrSyntax (quotation, term2)
@CrSyntax (maths, term3)
```

14.3.3 Galley definitions

In order to define a new galley several pieces of information are required — for example the procedure and font families to use, the device to use, and the styles to use. It is convenient to describe how to create all this information here, though other sections contain related material.

14.3.3.1 Styles objects

The style of a galley defines many of its global properties. In Mint a *styles object* can be declared and associated with galleys (thus making it easy to declare new galleys with properties similar to existing galleys). The following statement, extracted from file `DoverStyles.Mint`, declares the styles object `Style0` for device `Dover`.

```
@MakeStyle (dover, style0,
             0.75 pagewidth, 0 pagewidth, 0.025 pagewidth, 0 pagewidth,
             0.025 pagewidth, 0.025 pagewidth, 0.02 pageheight,
             0.01 pageheight, 0.02 pageheight, 0.01 pageheight,
             0.005 pageheight, 0.002 pageheight, 0.002 pageheight,
             0.025 pagewidth,
             True, True, True, False, Black, Transparent, ROr)
```

See section 4.1.2.1 for the meaning of each of the parameters.

14.3.3.2 Font families

A font family is declared using the `NewFontFamily` statement; after it has been declared fonts may be associated with it. This is done in two ways. First, *private* fonts are associated by means of the `AssocPrivF` statement; private fonts are used by Mint for the special symbols it needs (such as bullets and copyright symbols), and for mathematical typesetting. Second, the fonts that correspond to the standard face codes and font sizes are associated with the `AssocFont` statement. The following is from file `Fonts0Dover.Mint`.

```

@NewFontFamily (fonts0, pressfile)
@AssocPrivF (fonts0, mint, MintFontn)
@AssocFont (fonts0, n, r, TimesRoman10)
@AssocFont (fonts0, n, i, TimesRoman10i)
@AssocFont (fonts0, n, b, TimesRoman10b)
@AssocFont (fonts0, n, p, TimesRoman10bi)
@AssocFont (fonts0, n, c, CapsFont)
@AssocFont (fonts0, n, g, Hippo10)
@AssocFont (fonts0, n, z, ZFont10)
@AssocFont (fonts0, n, t, Gacha10)

```

The identifiers that may be used as second parameters to `AssocPrivF` are either `mint`, for the font used for special characters, or `maths`, for the mathematics extension font. See section 14.3.9 for the associations used for mathematical typesetting. (Of course, `mint`, `n`, `g`, etc, are themselves specified by `InitLexMap` statements.)

14.3.3 Procedure families

Procedure families are declared in a similar way to font families; once declared they can then have procedures associated with them. An entry in a procedure family specifies the box procedure, the environment parameters and the prefix to use for that particular environment. The following is an extract from file `Main0.Mint`.

```

@NewProcFamily (mainpf)
@AssocProc (mainpf, Centre, BoxStandard0, EnvCentre0, NoPrefix)
@AssocProc (mainpf, Display, BoxStandard0, EnvDisplay0, NoPrefix)
@AssocProc (mainpf, Verse, BoxStandard0, EnvVerse0, NoPrefix)
@AssocProc (mainpf, Maths, BoxMaths0, EnvMaths0, PostfixEqn)
@AssocProc (mainpf, MajorHeading, BoxStandard0, EnvMajorHeading0, NoPrefix)
@AssocProc (mainpf, Chapter, BoxSectionEnv0, EnvChapter0, PrefixChapter)
@AssocProc (mainpf, Section, BoxSectionEnv0, EnvSection0, PrefixSection)
@AssocProc (mainpf, Enumerate, BoxEnumerate0, EnvEnumerate0, NoPrefix)

```

14.3.4 Declaring galleys

Galleys are declared using the `NewGalley` statement. A galley requires a name (first parameter); a galley class which specifies how the galley will be used (second parameter); the font family, procedure family and style to be associated with the galley (third, fourth and fifth parameters); the environment whose box procedure will be invoked at the bottom of the interpreter stack (sixth parameter); and finally a boolean that specifies whether the galley is to start up immediately. The statements in `Manual10Galleys.Mint` are

```

@NewGalley (main, main, fonts0, mainpf, style0, mroot, true)
@NewGalley (footnotes, annotations, fonts1, footpf, style0, mfoot, true)
@NewGalley (annotations, annotations, fonts1, notepf, style0,
            manannotation, true)
@NewGalley (oddsandsods, miscellaneous, fonts2, oddspf, style0,
            mroot, true)

```

```
    moddsandsods, true)
@NewGalley (contents, contents, fonts0, contpf, style0, mcontents, false)
```

The allowed galley classes are as follows: **main**, **annotations**, **miscellaneous** and **contents**. There may be one or more **main** galleys but they must all have the same starting procedure; there can be any number of **annotations** galleys, and they are not restricted to having the same starting procedure; there can be only one **miscellaneous** galley; and the rules for **contents** galleys are the same as for **main** galleys.

14.3.3.5 Miscellaneous properties

The following statements are taken from **BasicDefs.Mint**; their actions can be understood from the rest of the Reference Manual.

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

@NewPattern (EmptyPattern)
@NewPattern (Line1, 1, 1, Black)

@NewBorderStyle (NoBorder, False, False, False, False,
                  EmptyPattern, EmptyPattern, EmptyPattern, EmptyPattern)
@NewBorderStyle (Width1, False, False, False, False,
                  Line1, Line1, Line1, Line1)
```

14.3.4 Font definitions

Mint allows very complete descriptions of the characters that occur in fonts; normally however it can produce pleasing output without these descriptions. The exception arises with mathematical typesetting, where full descriptions are needed. The state files contain information for typesetting using the Xerox Times Roman fonts; other fonts (such as those produced by Metafont) could use this information, but documents will look better if they use information specific to these fonts. This information can be extracted from the .TFM files in the case of fonts produced by Metafont.

The information is collected into several collections — *characteristics vectors* — that are named; these vectors are then associated with *font value objects*; finally font value objects are associated with particular fonts. In this way the same information can be reused.

14.3.4.1 Characteristics vectors

Each vector element is a 16-bit number, one for each character; packed into each number are a collection of indirect pointers to specific values that exist in a font value object; using a level of indirection allows the same characteristics vectors to describe several fonts. Character information has been grouped into the four vectors according to the likelihood with which the several items will be needed simultaneously. The information is:

CharBBI	IX0', IY0, IXX', IYY, ICL, ICR
CharBBX	X0, Y0, XX, YY
CharBBY	VX0', VY0, YRel, YT'
CharBBR	XRel, XT'

The primed quantities are related to the unprimed ones described in section 5.4.1 as follows:

$$\begin{aligned} XT' &= XT - XX \\ YT' &= YT - YY + Y0 - VY0 \\ IX0' &= IX0 - X0 \\ IXX' &= XX - IXX \\ VX0' &= VX0 + 0.5 * (XX - X0) \end{aligned}$$

The following is an extract from file **CharDefs.Mint**

```
@NewCharBBIs (StandardBBI)
@SetCharBBI (StandardBBI, @Char (#041), 0, 1, 0, 3, 1, 4)
@SetCharBBI (StandardBBI, @Char (#042), 0, 6, 0, 1, 1, 4)
@SetCharBBI (StandardBBI, @Char (#043), 0, 1, 0, 4, 1, 4)
@SetCharBBI (StandardBBI, @Char (#044), 0, 1, 0, 1, 1, 4)
@SetCharBBI (StandardBBI, @Char (#045), 0, 1, 0, 1, 1, 5)
```

and the following from file **MexDover.Mint**

```
@NewCharBBXs (MathsBBX)
@SetCharBBX (MathsBBX, @Char (#0), 0, 1, 0, 1)
@SetCharBBX (MathsBBX, @Char (#1), 0, 1, 0, 1)
@SetCharBBX (MathsBBX, @Char (#2), 0, 1, 0, 1)
@SetCharBBX (MathsBBX, @Char (#3), 0, 1, 0, 1)
@SetCharBBX (MathsBBX, @Char (#4), 0, 1, 0, 1)

@NewCharBBYs (MathsBBY)
@SetCharBBY (MathsBBY, @Char (#0), 0, 0, 1, 0)
@SetCharBBY (MathsBBY, @Char (#1), 0, 0, 1, 0)
@SetCharBBY (MathsBBY, @Char (#2), 0, 0, 1, 0)
@SetCharBBY (MathsBBY, @Char (#3), 0, 0, 1, 0)
@SetCharBBY (MathsBBY, @Char (#4), 0, 0, 1, 0)
```

14.3.4.2 Font value objects

The characteristics vectors describe very general appearances of fonts, such as the fact that a lower case *a* and *x* both sit on the base-line, and have the same height. For this reason the same characteristics could describe a wide range of font families (such as all Times Roman fonts and all Univers fonts, irrespective of their slope, weight or width). A *font value* object gives specific values to the indirect values specified by the characteristics vectors. For example it will describe the actual height of the *a* and *x*; moreover, these values are given in terms of font-relative units (such as *ems*), which in Mint need not be the same size in horizontal and vertical directions. For this reason a font value object can describe a family of fonts (for example, all Univers fonts can be described to a reasonable degree of accuracy by the same font value object).

Font value objects are defined in two steps. First, an object is declared using the **NewFontValue** statement, and then specific values are given to it by the **SetFontValue** statement, which takes the name of a characteristic, such as *Y0*, and an indirect pointer number, and the size of the value.

In the state files three font value objects are defined — that for normal and italic Times Roman fonts, that for the mathematics symbol fonts, and that for the mathematics extension font. Extracts from the files **CharDefs.Mint**, **MathsDefs.Mint** and **MexDover.Mint** follow.

```
@NewFontValues (StandardR)
@SetFontValue (StandardR, IY0, 1, 1.00em)
@SetFontValue (StandardR, IY0, 2, 1.05em)
@SetFontValue (StandardR, IYY, 1, 0.00em)
@SetFontValue (StandardR, IYY, 2, 0.30em)
@SetFontValue (StandardR, ICL, 1, 0.00em)
@SetFontValue (StandardR, ICR, 1, 0.00em)

@NewFontValues (StandardS)
@SetFontValue (StandardS, Y0, 1, -0.25em)
@SetFontValue (StandardS, Y0, 2, -1.25em)
@SetFontValue (StandardS, YY, 1, 1.00em)
@SetFontValue (StandardS, YY, 2, 0.00em)

@NewFontValues (StandardM)
@SetFontValue (StandardM, Y0, 1, -4.23em)
@SetFontValue (StandardM, Y0, 2, -2.12em)
@SetFontValue (StandardM, Y0, 3, -6.35em)
@SetFontValue (StandardM, YY, 1, 0.00em)
@SetFontValue (StandardM, YY, 2, 2.64em)
@SetFontValue (StandardM, YY, 3, 0.39em)
```

14.3.4.3 Font information

The final stage of associating information with a font is to specify the characteristics vectors, the font value object, and several other items of information (not all of which are used by Mint at present). The following information is from file **MexDover.Mint**, which specifies the information for performing

mathematical typesetting using the TimesRoman fonts; information is not associated with other fonts since they are used only for normal text, and Mint's defaults are sufficiently good to produce reasonable-quality output²².

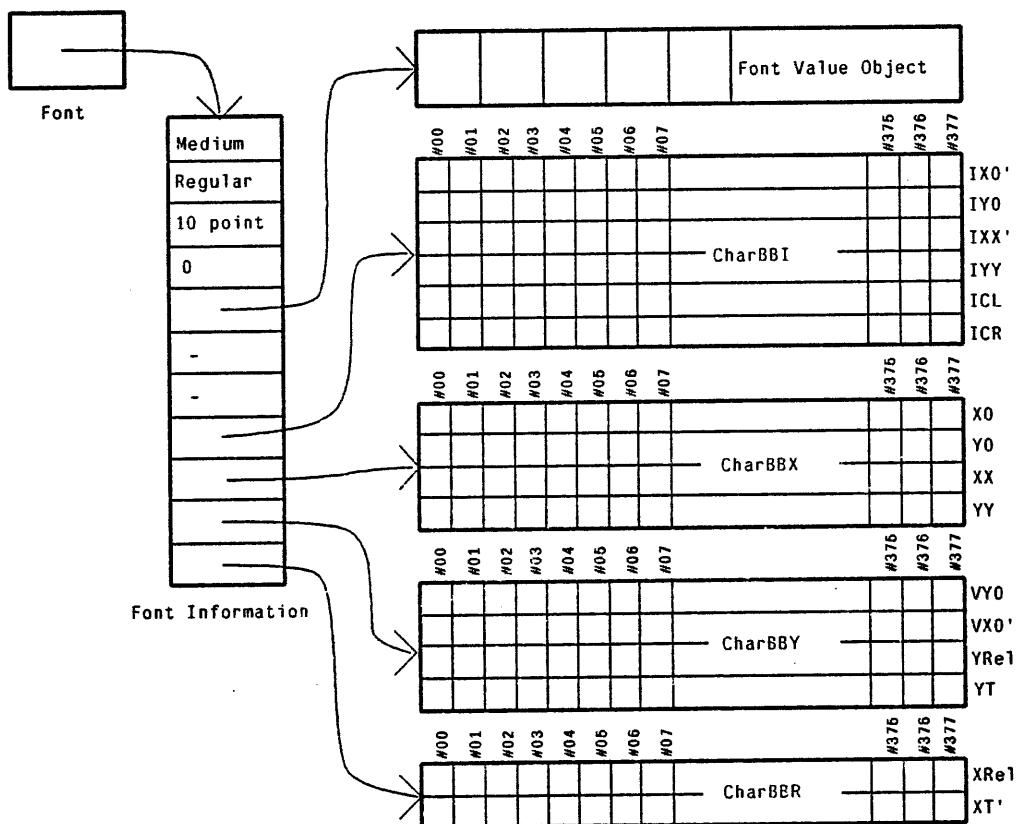


Figure 8. Font information structures

```

@SetFontInfo (PressFile, CMathX10s10, Medium, Regular, 10 point, 0,
               StandardM, 100 micas, 100 micas, MathsBBX, MathsBBY)

@SetFontInfo (PressFile, TimesRoman10, Medium, Regular, 10 point, 0,
               StandardR, , , StandardBBI)
@SetFontInfo (PressFile, TimesRoman7, Medium, Regular, 7 point, 0,
               StandardR, , , StandardBBI)
@SetFontInfo (PressFile, TimesRoman6, Medium, Regular, 6 point, 0,
               StandardR, , , StandardBBI)
@SetFontInfo (PressFile, TimesRoman10i, Medium, Regular, 10 point, 0.16,
               StandardR, , , StandardBBI)
@SetFontInfo (PressFile, TimesRoman7i, Medium, Regular, 7 point, 0.16,
               StandardR, , , StandardBBI)
@SetFontInfo (PressFile, TimesRoman6i, Medium, Regular, 6 point, 0.16,
               StandardR, , , StandardBBI)

```

²² However, the SetSpacing statement needs additional font information. The .Mss file for this section gives an example.

```
    StandardR, , , StandardBBI)
@SetFontInfo (PressFile, CMSy10s10, Medium, Regular, 10 point, 0,
               StandardS, 8 points, 8 points, , SymbolBBX)
@SetFontInfo (PressFile, CMSy7s7, Medium, Regular, 7 point, 0,
               StandardS, 5.6 points, 5.6 points, , SymbolBBX)
@SetFontInfo (PressFile, CMSy6s6, Medium, Regular, 6 point, 0,
               StandardS, 4.8 points, 4.8 points, , SymbolBBX)
```

The third, fourth and fifth parameters specify the weight, width and nominal point-size; this information is not used by Mint at present. The sixth parameter is the slope of the font, in radians measured clockwise from the vertical; this is used to calculate the displacement of accents and the size of italic corrections. The seventh parameter is the font value object that gives the specific values of the font parameters. The eighth and ninth are the sizes for the *em* in the *X* and *Y* directions — if omitted the actual size of the character at position #115 is used. The remaining parameters are the characteristics to be used. Mint has a sequence of defaults to determine values for character parameters if information is not provided.

14.3.5 Counter and conversion definitions

Counters enter into the state files in two ways. First, they need to be declared with their associated parent counters, starting values and increments; second, they are used in prefixes. Over the first you have full control, but over the second most of the decisions are buried inside prefix declarations built into Mint. Some day it may become possible to specify prefixes at the .Mss level.

14.3.5.1 Registering conversions and counters

The conversions are built-in; identifiers are associated with the basic conversions using the **StdConv** statement (from file **BasicDefs.Mint**; conversion 0 is required for internal reasons):

```
@StdConv (0)
@StdConv (1, RomanLC)
@StdConv (2, RomanUC)
@StdConv (3, Arabic)
@StdConv (4, AlphaLC)
@StdConv (5, AlphaUC)
@StdConv (6, LetterLC)
@StdConv (7, LetterUC)
@StdConv (8, Binding)
```

The statement **Register** is used for registering conversion styles and counters that are used in prefixes before they are declared (this is the case with all the standard conversion styles and counters, but is unlikely to be the case for other conversions and counters declared by the user). The following are in file **OtherDefs.Mint**.

```
@Register (annotestyle, 0)
@Register (partstyle, 0)
```

```

@Register (pagestyle, 0)
@Register (chapterstyle, 0)

@Register (ChapterNo, 1)
@Register (SectionNo, 1)
@Register (SubSectionNo, 1)
@Register (ParagraphNo, 1)

@Register (PartNo, 2)
@Register (PageNo, 3)
@Register (AnnoteNo, 4)
@Register (EquationNo, 5)

```

A second parameter of 0 is used for the non-primitive styles, 1 is used for counters, and values greater than 1 are used for pseudo-counters. The meanings of these values are built into Mint.

14.3.5.2 Declaring counters

Counters that do not appear in prefixes do not need to be registered. However, they all need to be declared. A declaration specifies the parent counter, the starting value and the increment:

```

@NewCounter (PartNo, , 1, 1)
@NewCounter (PageNo, PartNo, 1, 1)
@NewCounter (EquationNo, ChapterNo, 1, 1)
@NewCounter (ChapterNo, , 1, 1)
@NewCounter (SectionNo, ChapterNo, 1, 1)
@NewCounter (SubSectionNo, SectionNo, 1, 1)
@NewCounter (ParagraphNo, SubSectionNo, 1, 1)

```

These examples come from file `Manual10Defs.Mint`.

14.3.5.3 Associating conversions

For each style that will be used in the document, a basic style must be associated using the `AssocConv` statement. The following statements come from file `Manual10Defs.Mint`.

```

@AssocConv (AnnoteStyle, Arabic)
@AssocConv (PartStyle, Arabic)
@AssocConv (PageStyle, Arabic)
@AssocConv (PlaceStyle, Arabic)
@AssocConv (ChapterStyle, AlphaUC)
@AssocConv (SectionStyle, Arabic)
@AssocConv (SubSectionStyle, Arabic)
@AssocConv (ParagraphStyle, Arabic)

```

14.3.5.4 Prefixes

There are seven styles of prefix. All of them are created using the `MakePrefix` statement. The first parameter is the identifier for the prefix, and the second is the style of the prefix. The rest of the parameters are used in different ways by the different prefixes; they are:

- | | |
|---------|--|
| Style 0 | Declares a null prefix. |
| Style 1 | Declares a prefix that will centre a heading; the prefix that creates the heading for each major section of this manual is defined by a prefix in this style. The third parameter is the counter to be used, the fourth parameter is the conversion style for the counter, and the fifth parameter is the lexeme that precedes the number (in the case of this manual, which is form 1, the lexeme is <code>Part</code>). |
| Style 2 | Declares the standard section heading prefix. The third parameter is the counter that gets incremented whenever such a prefix is used, and the fourth parameter is the conversion style. |
| Style 3 | Declares the prefix used for tables and figures. The third parameter is the counter that is used and incremented, the fourth parameter is the conversion style, and the fifth parameter is the lexeme that precedes the counter. |
| Style 4 | Declares the prefix that generates the copyright notice and year. The third parameter is the counter to be used, the fourth parameter is the conversion style for the counter, and the fifth parameter is the lexeme that precedes the copyright symbol. |
| Style 5 | Declares the postfix that generates the equation number for equations. The third parameter is the conversion style. |
| Style 6 | Declares a prefix for an annotation. The third parameter is the lexeme that will precede the annotation. |

The following statements come from file `Manual10Defs.Mint`.

```
@MakePrefix (PrefixChapter, 1, ChapterNo, ChapterStyle, Chapter)
@MakePrefix (PrefixSection, 2, SectionNo, PlaceStyle)
@MakePrefix (PrefixSubSection, 2, SubSectionNo, PlaceStyle)
@MakePrefix (PrefixParagraph, 2, ParagraphNo, PlaceStyle)
@MakePrefix (PrefixFigure, 3, FigureNo, FigureStyle, Figure)
@MakePrefix (PrefixTable, 3, TableNo, TableStyle, Table)
@MakePrefix (PrefixCopyrtN, 4, CopyRtNo, CopyRtStyle, Copyright)
@MakePrefix (PostfixEqn, 5, EquationStyle)
```

The prefixes are associated with galleys through procedure families.

14.3.5.5 Place conversions

Pseudo-counters generate a sequence of counter values, rather than a single one; there needs to be some more control on the appearance of these values than that provided by the conversions. For example, the `PlaceConv` style is set to be `Arabic` in the `Manual` document type, but for an appendix section, the place reference should appear as `A.5`, not `1.5`. The statement

```
@ApplTrans (AppendixNo, LetterUC)
```

specifies that the `AppendixNo` counter in a `placestyle` conversion should differ from the rest.

Finally, when place conversion is done, Mint needs to know where in the family of counters it should stop searching for descendent counters. Thus if you declare (for example) a `subparagraph` counter that counts within the `paragraph` counter, but you do not want it to appear when you should use the `placestyle` conversion, you use the `NewPCS` statement. The following statements are in `Manual0Defs.Mint`.

```
@NewPCS (ParagraphNo)
@NewPCS (AppendixSecNo)
```

14.3.5.6 Cross reference

When a dominating environment occurs within another environment, Mint needs to leave a cross reference within the latter environment. This can be invisible (as is the case when a page command occurs in a manuscript), or it can be visible (as is the case when a footnote occurs in a manuscript). These cross references are declared using `MakeCRef`; the statement takes several parameters, the first of which is an indication of the style of cross-reference, and the second is the identifier for the cross reference. There are three such styles:

- | | |
|---------|--|
| Style 0 | This declares the null cross reference. |
| Style 1 | This declares the invisible cross reference used by page commands. |
| Style 2 | This declares a footnote cross reference. The counter associated with the cross reference is given by the third parameter. |

The following is taken from file `Manual0Defs.Mint`.

```
@MakeCRef (0)
@MakeCRef (1, crpageactn)
@MakeCRef (2, crannote, annotestyle)
```

The null cross reference must occur first.

14.3.6 Presentation definitions

14.3.6.1 Page styles

Page styles are associated with box environments through the use of the `PageStyle` environment parameter. This parameter specifies which entry in the layout vector of a presentation will be used for creating the pages for this environment. Two page styles, `skip` and `default`, play a special role in page layout, as follows. A page style may specify that the current box, and all that follow it that have `skip` for their page style, are to be laid out using the same routine; such a page style is *continuing*. Alternatively, a page style can specify that it is not continuing, in which case only the current box and those nested within it that have `skip` for their page styles will be converted using the specified layout. Examples of these two cases are the `chapter` environment, which specifies the page style `default` (which is continuing), and forces all boxes that follow to be laid out using the same layout; and the `titlepage` environment, which only controls the layout of environments nested within it. If a non-continuing page layout is followed by an environment that specifies `skip`, then a layout of `default` is assumed.

The following are the page style declarations in `PageDefs.Mint`:

```
@NewPageStyle (Skip, True)
@NewPageStyle (Default, True)
@NewPageStyle (TitlePage, False)
@NewPageStyle (Contents, False)
```

The second parameter specifies whether the page style is continuing or not.

14.3.6.2 Presentations

Presentations map between page styles and layouts, and provide the data structures — the parts — that contain the finished pages. Parts are associated with a presentation using the `AssocPart` statement; the second parameter is the name of the part. The following are from file `PageDefs.Mint`:

```
@NewPresentation (Standard)
@AssocPart (Standard, TitlePage)
@AssocPart (Standard, Contents)
@AssocPart (Standard, MainBody)
```

The printing routines that are invoked by the Mint command interface simply print the pages in each of the parts in the order in which the parts are associated with the presentation.

14.3.6.3 Layouts

A layout has to be associated with each entry of the layout vector. The vector contains an entry for each page style except for `skip`. A layout comprises an indication of a layout routine, and the parameters that will get passed to it when it is invoked. There are four layout routines; each has a different statement to specify the parameters. For example `NewDefault` is used to create a layout that will use the `Default` layout routine. In general two sorts of parameter can be passed to the layout routines — those that have a global effect, like `FinishOnEven`, and those that control the appearance of the page areas. Parameters of the second sort are collected up into *page area parameters*, which are objects that can be declared. More details are given in section 8.2.

No page area parameters are used by the standard layouts; however, a null set of page area parameters must be declared.

To associate a layout with the layout vector of some presentation, it is necessary to specify which part the pages will get sent to by using the `AssocLayout` statement. The first and second parameters specify the presentation and the part, the third parameter specifies the page style, and the fourth parameter specifies the layout that will be used to create the pages.

The following are from `PageDefs.Mint`.

```
@NewAreaParams ()

@NewTitlePage (TLayout, True, ., .)
@NewContents (CLayout, True, ., ., False, False)
@NewDefault (DLayout, True, ., ., False, False)
@NewPasteup (PLayout, True, ., ., False)

@AssocLayout (Standard, TitlePage, TitlePage, TLayout)
@AssocLayout (Standard, Contents, Contents, CLayout)
@AssocLayout (Standard, MainBody, Default, DLayout)
```

14.3.7 Bibliographies, contents definitions and indexes

Several bibliographies are declared in the standard state files, however it is the user's responsibility to ensure that the correct set of bibliography macros are used to lay out the bibliography. The following occur in `BasicDefs.Mint`

```
@NewBib (StdNumeric,
        @"" "@begin(r)[", @Char (',) @Char (Sp), @"""]@end(r)",
        KeyWord, Numeric, **)
@NewBib (StdAlphabetic,
        @"" "@begin(r)[", @Char (',) @Char (Sp), @"""]@end(r)>",
        KeyWord, Alphabetic, *****)
@NewBib (CACM,
```

```
@""@begin(r)[", @Char (',)@Char (Sp), @"""]@end(r)",  
KeyWord, Numeric, **)  
@NewBib (IEEE,  
        @""@begin(r)@begin(+)", @Char (',), @""@end(+}@end(r)",  
        CiteOrder, Numeric, **)
```

The first parameter is the bibliography identifier — this occurs in the `bibstyle` parameter in the `make` statement. The next three parameters specify the starting string, the separating string, and the terminating string in citations. The fifth and sixth parameter specify the way the citations will be sorted into the bibliography, and the way the citations will appear. The last parameter gives the filler lexeme that will be placed into the slugs before the citation is known.

Several conditions need to be satisfied for tables of contents to be created. First, the document type must have a galley whose type is `Contents`; this is the case for document types `manual`, `report` and `thesis`. Second, the presentation must have an entry in the `contents` position of the layout vector; this is the case for the document types above. Third, `contents tables` must have been defined: the statements below show how to do this. Finally, macros definitions must be available: the `contentsmacros` library supplies suitable ones.

Contents tables are declared using the `NewContentsTable` statement. Each contents table will appear on a new page (possibly an odd numbered page, depending on the value of `FinishOnEven`). The first parameter is the name of the table; the second parameter is non-null if the entry in the table is to be the text of the caption within an environment, rather than the text of the environment itself. The following are from file `BasicDefs.Mint`.

```
@NewContentsTable (Figures, True)  
@NewContentsTable (Tables, True)  
@NewContentsTable (Sections)
```

After a contents table has been defined, environments are associated with it. The text from these environments, or from the captions within them, will be used to create the table of contents. The following occur in `Manual10Defs.Mint`.

```
@AssocContents (Figures, Figure)  
@AssocContents (Tables, Table)  
@AssocContents (Sections, Chapter)  
@AssocContents (Sections, Section)  
@AssocContents (Sections, SubSection)  
@AssocContents (Sections, Paragraph)  
@AssocContents (Sections, Appendix)  
@AssocContents (Sections, AppendixSection)
```

The mechanism that is used to create the tables is as follows (illustrated for the `Sections` table). Mint first calls a macro `begsections`, with no parameters; then it calls macros `tcchapter`, `tcsection`, etc,

with a label parameter and the text of the environment²³ for each environment that is to appear in the table; finally Mint calls a parameterless macro `endsections`. The macro definitions in the library file `contentsmacros` illustrate how to use these calls.

Currently there are no statements in the state files to help create indexes.

14.3.8 Device definitions

Mint specifies the characteristics of devices using a hierarchy of definitions. At the lowest level are *device drivers* and *font formats*. Mint's understanding of these is built-in, and major programming is needed to add more drivers and font formats. At the next level *device classes* are defined. A device class corresponds to a collection of devices, all using the same driver and same font representation, and having the same raster resolutions. Finally specific devices are declared. A device gives specific values to some characteristics, in particular the size of the page, and in addition is associated with specific crossproofing fonts (that *may* be a mistake).

It is unlikely that any driver will be able to handle any more than one font format, so users don't get much freedom to define their own device classes. The statement `NewDClass` specifies the name of the device class, the driver and format, the resolutions in the *X* and *Y* directions (in rasters, pixels, or whatever it is that the device driver understands, per basic unit of length, whatever that has been chosen to be, see section 14.3.1.3), the internal scaling factor (used to avoid 16-bit overflow), and the raster drawing operations available to the device, expressed as a bit string. The following are from `BasicDefs.Mint`.

```
@NewDCClass (PerqScreen, PerqDriver, KstFormat, 90, 96, 1, #377)
@NewDCClass (PressFile, PressDriver, XeroxFormat, 2540, 2540, 5, #20)
```

Devices can be declared at any time (in the Sapphire version they are declared automatically when you change the document window size). A device declaration gives an identifier, which appears in the crossproofing statements, and may be used interactively while the presentation is being viewed, and also binds the size of the page. Mint requires no modification to make use of devices with different page sizes (e.g. large flat-bed plotters, A4 paper, landscape displays). The following are from `BasicDefs.Mint`.

```
@NewDevice (Perq, PerqScreen, 8.4 in, 9.5 in)
@NewDevice (Dover, PressFile, 8.5 in, 11 in)
```

²³ Footnotes are stripped from it, of course.

14.3.9 Mathematical definitions

Most of the fine-structure of the mathematical typesetting facilities are obtained through statements that are in the state files. Should the facilities I have provided not prove rich enough, it is likely that you will need to alter these statements. In particular, the appearance of the mathematical typesetting has been tuned for a particular set of fonts (a non-standard blend of Times Roman and Metafont fonts) and if you chose to use other fonts you will probably wish to re-tune the parameters.

14.3.9.1 Composite symbols

The large brackets are created by combining several separate glyphs. For example, a large brace consists of a top piece (1), several repetitions of an extension piece (1), a middle part (1), several more repetitions of the extension part (1), and finally a bottom part (1). Such a brace can be described by the finite state expression AB*CB*D. Mint uses such expressions, together with knowledge of which glyphs correspond to A, B, etc, to create the brackets. The `MathsVCA` statement is used to describe such glyphs. The first parameter is a small number that identifies the glyph (this number gets used in the `paren`, `brace`, etc, statements in the maths environments), the second is a small number that specifies the finite state description of the glyph, then follow four glyphs (which will come from the maths extension font) that go to make up the glyph, followed by some number of glyphs, which are single character representations of the glyph, in increasing size. For example, 8 is an open parenthesis, 9 is the finite state expression AB*C for vertical glyphs (there's another value for horizontal glyphs), characters #100, #102 and #60 are the bottom part, middle extension and top part of large open parentheses, and characters #0, #20, #22 and #40 are single character open parentheses of increasing size.

The following is from `MathsDefs.Mint`:

```
@MathsVCA (7, 0)
@MathsVCA (8, 9, @Char (#100), @Char (#102), @Char (#60), @Char (#377),
             @Char (#0), @Char (#20), @Char (#22), @Char (#40))
@MathsVCA (9, 9, @Char (#101), @Char (#103), @Char (#61), @Char (#377),
             @Char (#1), @Char (#21), @Char (#23), @Char (#41))
@MathsVCA (10, 10, @Char (#72), @Char (#76), @Char (#74), @Char (#70),
             @Char (#10), @Char (#32), @Char (#50))
```

14.3.9.2 Spacings

To define the spacings that occur between symbols, two steps are taken. First, for each ordered pair of symbol types you have to specify symbolically the spacing that is needed between them; this is done using the statement `SetSpType`. Then you have to give the size of each spacing type using the statement `SetSpValue`, for each of the font sizes n, s and ss, in terms of thousandths of an em — the absolute size of the em is determined from the font characteristics of the m0 font used for the current maths mode. The following come from `MathsDefs.Mint`.

```

@SetSpType (Ord, Ord, Thin)
@SetSpType (Ord, Ord1, Thin)
@SetSpType (Ord, Num, Thin)
@SetSpType (Ord, Op, Thin)
@SetSpType (Ord, BinOp, Op)
@SetSpType (Ord, RelOp, Thick)
@SetSpType (Ord, Word, Fat)
@SetSpType (Ord1, Ord, Thin)
@SetSpType (Ord1, Ord1, Hair)
@SetSpType (Ord1, Num, Thin)

@SatSpValue (Thin, n, 167)
@SetSpValue (Thin, s, 167)
@SetSpValue (Thin, ss, 167)
@SetSpValue (Control, n, 222)
@SetSpValue (Control, s, 167)
@SetSpValue (Control, ss, 167)
@SetSpValue (Op, n, 222)
@SetSpValue (Op, s, 67)
@SetSpValue (Op, ss, 67)

```

In addition to the inter-symbol spacings, there are other values that are used by the mathematical typesetting parts of Mint. The `MathsParams` statement sets these values. See section 12.3.6 for what they do.

```
@MathsParams (0.30em, 0.07em, 0.14em, 0.05em, 0.17em, 0.14em, 0.09em,
-0.10em, -0.09em, -0.06em, 0.04em, 0.04em, 175)
```

14.3.9.3 Standard symbols

I have included a few common definitions in the file `MathsDefs.Mint`. They aren't intended to be exhaustive, and they aren't representative either. Anyone who wants to come up with a short list of definitions should let me know. The following are some of the definitions.

```

@MDef (@tilde, @Char (#176), Over, M0)
@MDef (sin, sin, Op, M0)
@MDef (cos, cos, Op, M0)
@MDef (+, +, BinOp, M0)
@MDef (-, @Char (#30), BinOp, M0)
@MDef (*, @Char (#02), BinOp, M2)
@MDef (le, @Char (#24), RelOp, M2)
@MDef (ge, @Char (#25), RelOp, M2)
@MDef (ne, @Char (#34), RelOp, M2)
@MDef (<, <, RelOp, M0)
@MDef (>, >, RelOp, M0)
@MDef ((, (, Open, M0)
@MDef (;, ;, Punct, M0)
@MDef (inf, @Char (#61), Ord, M2)

```

14.3.9.4 Mathematical fonts

The following are the font associations; they are extracted from file `mfonts0dover.mint`.

```
@AssocFont (Fonts0, n, m0, TimesRoman10)
@AssocFont (Fonts0, n, m1, TimesRoman10i)
@AssocFont (Fonts0, n, m2, CMSy10S10)
@AssocFont (Fonts0, s, m0, TimesRoman7)
@AssocFont (Fonts0, s, m1, TimesRoman7i)
@AssocFont (Fonts0, s, m2, CMSy7S7)
@AssocFont (Fonts0, ss, m0, TimesRoman6)
@AssocFont (Fonts0, ss, m1, TimesRoman6i)
@AssocFont (Fonts0, ss, m2, CMSy6S6)

@AssocPrivF (Fonts0, maths, CMathX10S10)
```

14.3.10 State file definitions

The first three parameters to a `make` statement are simply concatenated together to yield a state file name that is then read; there is nothing intrinsic in these parameters that specifies that the state file will be appropriate for the document type, form and device given²⁴, nor are the document type, form number and device recorded in the file automatically. The statement `SetDumpP` specifies the values that will get returned by `@value(documenttype)`, `@value(form)` and `@value(device)`. This statement occurs in file `Manual0Dover.Mint`.

```
@SetDumpP (manual, 0, dover)
```

²⁴ Of course, all the standard state files that I have created do conform to expectations.