Copyright
Copyright © 1999, Ian Clatworthy
All rights reserved.
You are free to copy and distribute this manual as often as you like. If parts of this manual are included in commercial products such as books and magazine articles, credit must be given to the author.

Trademarks
FrameMaker, FrameViewer, Acrobat, PDF and PostScript are registered trademarks of Adobe Systems Incorporated.
Windows is a registered trademark of Microsoft Corporation.
Unix is a registered trademark of X/Open Company Ltd.
Information Mapping is a registered trademark of Information Mapping, Incorporated.
Other trademarks are the property of their respective owners.

Tools
This document was generated using SDF 2.001.
# Table of Contents

About This Manual ............................................................... v

1 Getting Started .............................................................. 1-1
   Introducing SDF ......................................................... 1-1
   Installing SDF .......................................................... 1-3
   A simple example ....................................................... 1-8
   Generating outputs ..................................................... 1-11
   Document styles ......................................................... 1-13
   Building a title ........................................................ 1-15

2 Language overview ......................................................... 2-1
   Basic concepts ........................................................ 2-1
   Paragraphs .............................................................. 2-2
   Phrases ................................................................. 2-5
   Macros ................................................................. 2-7
   Variables .............................................................. 2-8
   Filters ................................................................. 2-11
   Attributes ............................................................ 2-13
   Expressions ........................................................... 2-14

3 Paragraphs ........................................................................... 3-1
   Headings ................................................................. 3-1
   Example Paragraphs ..................................................... 3-3
   Lists ........................................................................... 3-6
   Notes ........................................................................... 3-9

4 Figures .............................................................................. 4-1
   Overview ...................................................................... 4-1
   Creating Figures ......................................................... 4-3
   Adding Figure Titles .................................................... 4-5
   Positioning a Figure ..................................................... 4-6

5 Tables ............................................................................... 5-1
   Overview ...................................................................... 5-1
   Table Styles ............................................................... 5-3
   Adding Table Titles ....................................................... 5-5
   Sizing Columns ........................................................... 5-6
   Positioning Tables ......................................................... 5-10
   Creating Heading, Footing and Group Rows ...................... 5-11
   Other Table Goodies ...................................................... 5-13
   Spreadsheet calculations .............................................. 5-18

6 SDF Advanced Features ..................................................... 6-1
   Introduction ............................................................ 6-1
About This Manual

**Purpose**
This manual provides an introduction to SDF - a documentation system for software developers.

**Scope**
This manual contains a tutorial-style introduction to the commonly used features in SDF.

The following documents may also be of interest:

- *The SDF Document Development System* - an overview of the architecture and features
- *SDF Quick Reference* - a concise summary of the most frequently used SDF features
- *SDF Reference* - detailed reference material on all SDF features
- *SDF Guru Guide* - information on extending SDF.

**Amendments**
Suggested enhancements and corrections to SDF and its documentation can be forwarded to sdf-bugs@mincom.com.

**Note:** This manual documents version 2.001 of SDF.
Chapter 1. Getting Started

Introducing SDF

What is SDF?

SDF (Simple Document Format) is a freely available document development system which generates high quality outputs in a variety of formats from a single source. The output formats supported include PostScript™, HTML, POD, plain text, man pages, LaTeX, SGML, Windows™ help, MIF and RTF. If the idea of specifying documents in a logical manner via a simple markup language sounds appealing, SDF may be useful to you.

SDF documents are simple to create and maintain, minimising the time spent on documentation. In particular, SDF directly supports the creation and maintenance of large, on-line documentation systems (including intranets) via centralised hypertext management and rule-based hypertext generation.

What is SDF useful for?

As well as normal documents, SDF is useful for:

- user manuals (paper-based and online)
- online document catalogs
- Perl module documentation
- Delphi™ component documentation
- source code listings (pretty printing most languages).

SDF is also good for literate programming, i.e. documentation can be embedded in the comments of most programming languages and can be selectively extracted. The embedded documentation can be in any markup language you like, although there are advantages to using SDF’s markup language.

Requirements

SDF has been completely developed in Perl, a popular and highly portable scripting language. Perl 5.004 or later is recommended, although SDF may run on earlier versions of Perl 5.

To generate PostScript, SDF requires one of the following:

- the freely available pod2ps program
- the freely available SGML-Tools and LaTeX packages
• FrameMaker 5.x or later
• another word processor which can import RTF.

Earlier versions of FrameMaker will generally work, but I don't explicitly support them.

To generate Windows help, a help compiler (e.g. hcp.exe) is required.

SDF on the World Wide Web
The SDF home page URL is http://www.mincom.com/mtr/sdf. From the home page, you can:
• download the latest software
• browse the FAQ
• report defects
• suggest enhancements.

Fetching SDF
SDF is available using anonymous ftp. The URL of the base directory is ftp://ftp.mincom.com/pub/mtr/sdf. The latest release is available in a variety of formats.

Mailing lists
The following mailing lists are available:
• sdf-users@mincom.com - for general questions
• sdf-bugs@mincom.com - for reporting bugs and fixes.

To subscribe to these lists, send email to sdf-users-request@mincom.com and/or sdf-bugs-request@mincom.com for instructions on using factotum, the majordomo variant that manages these lists. In short, send email to factotum@mincom.com with a message body of subscribe sdf-users or subscribe sdf-bugs.
Installing SDF

Overview

If SDF is not yet installed, you will need to fetch it and install it. The latest version of SDF can be fetched from the SDF download page, http://www.mincom.com/mtr/sdf/download.html.

If Perl is not installed, you will also need to fetch and install it. Various ports of Perl can be obtained from the Perl Home Page (http://www.perl.com/index/).

SDF installation details are provided below for:

- Unix
- Windows NT
- Windows 95
- MS-DOS/Windows 3.x.

Some general Installation Notes are also provided.

Unix

Use the `tar` and `gunzip` programs to unpack the distribution like this:

```
$ gunzip sdf-2.000.tar.gz
$ tar xvf sdf-2.000.tar
```

Build, test and install the software like this:

```
$ perl Makefile.PL
$ make
$ make test
$ make install
```

Windows NT

Unpack the distribution using a program which understands long file names (e.g. WinZip).

Within the Control Panel, go to the System properties, select the Environment tab and edit the variables as follows:

1. Update the PATH to include the `bin\dos` directory where SDF is installed.
   
   For example, if SDF is installed in `c:\sdf`, then you need to edit the PATH system variable to include `c:\sdf\bin\dos`.

2. Edit (or add) the PERL5LIB variable to include the Perl libraries used by SDF.
For example, if SDF is installed in `c:sdf`, then you need to edit the PERL5LIB variable to include `c:sdf\perllib`.

3. Set the SDFBIN variable to point to the directory containing SDF’s scripts, including the trailing `\`.

   For example, if SDF is installed in `c:sdf`, then you need to set the SDFBIN variable to `c:sdf\bin\`.

Test the software. To do this:

1. Start an MS-DOS window.
2. Change directory to the `t` directory within the SDF distribution.
3. Run `sdf` without any arguments to confirm that SDF was successfully installed.
4. Run `sdftest`.

If anything fails, `sdftest` will leave the failed `.out` and `.log` files in the relevant directories.

**Windows 95**

Unpack the distribution using a program which understands long file names (e.g. WinZip).

Update the PATH to include the `bin\dos` directory where SDF is installed. For example, if SDF is installed in `c:sdf`, then the command is:

```
set PATH=%PATH%;c:sdf\bin\dos
```

Edit (or add) the PERL5LIB system variable to include the Perl libraries used by SDF. For example, if SDF is installed in `c:sdf`, then the command is:

```
set PERL5LIB=%PERL5LIB%;c:sdf\perllib
```

Set the SDFBIN variable to point to the directory containing SDF’s scripts, including the trailing `\`. For example, if SDF is installed in `c:sdf`, then the command is:

```
set SDFBIN=c:sdf\bin\n
```

Test the software. To do this:

1. Change directory to the `t` directory within the SDF distribution.
2. Run `sdf` without any arguments to confirm that SDF was successfully installed.
3. Run `sdftest`.

If anything fails, `sdftest` will leave the failed `.out` and `.log` files in the relevant directories.

**Note:** PATH, PERL5LIB and SDFBIN should be set within your `autoexec.bat` file.

---

**MS-DOS/Windows 3.x**

Use `pkunzip` to unpack the distribution. The command is:

```
pkunzip -d sdf.zip
```

Update the PATH to include the `bin\dos` directory where SDF is installed. For example, if SDF is installed in `c:\sdf`, then the command is:

```
set PATH=%PATH%;c:\sdf\bin\dos
```

Edit (or add) the PERL5LIB system variable to include the Perl libraries used by SDF. For example, if SDF is installed in `c:\sdf`, then the command is:

```
set PERL5LIB=%PERL5LIB%;c:\sdf\perllib
```

Set the SDFBIN variable to point to the directory containing SDF’s scripts, including the trailing \. For example, if SDF is installed in `c:\sdf`, then the command is:

```
set SDFBIN=c:\sdf\bin\ 
```

Test the software. To do this:

1. Change directory to the `t` directory within the SDF distribution.
2. Run `sdf` without any arguments to confirm that SDF was successfully installed.
3. Run `sdftest`.

If anything fails, `sdftest` will leave the failed `.out` and `.log` files in the relevant directories.

**Note:** PATH, PERL5LIB and SDFBIN should be set within your `autoexec.bat` file.
Installation Notes

Distribution Formats
The tar.gz and zip distributions should both contain the same set of files. Therefore, it shouldn't matter which distribution you obtain provided you have the necessary software to unpack it.

The sdf.ini File
The sdf.ini configuration file (in the sdf/home directory on Perl's library path) contains several sets of configuration parameters which you may want to edit after SDF is installed. Refer to the comments in the file for further details.

Win32 Support
SDF has been briefly tested with:

- Gurusamy Sarathy's 5.004_02 Perl port (August 97), and
- ActiveState's Perl port build 315 (based on 5.003_07).

Both seem to work ok, although running sdftest with the 5.004_02 port leaves zero length .out and .log files for the tests which work (when they should be deleted).

The SDFBIN Environment Variable
This environment variable is only used by the Windows/DOS wrapper batch files within bin/dos to find the real scripts. Therefore, if you wish to wrap the scripts in some other way, then there is no need to set this variable.

Macintosh support
Unlike earlier versions, the 2.000 release includes support for Macintosh filename handling. Installation instructions for the Macintosh will be made available soon.

Testing Notes
When the sdf command is run without arguments, the output should be something like:

```
```

© Ian Clatworthy
v[verbose]] [-w width] sdf_file
...
purpose: convert an sdf file to another format
version: 2.000    (SDF 2.000)

If this fails, check the following:
1. perl 5.003 or later is installed (type perl -v to test)
2. the sdf script is on your path and under Unix, is executable.

SDF is supplied with a regression testing system. To run the tests, change
to the t directory within the SDF distribution and execute sdftest like
this:
   sdftest

This command can be used at anytime to verify your installation of SDF.

The SDF Sample Showcase contains SDF source code and matching
outputs. If you wish to conduct further tests, these samples can be used.
A simple example

A sample SDF document is shown below:

```sdf
!define DOC_NAME "GalaxyBuilder"
!define DOC_TYPE "Discussion Paper"
!define DOC_AUTHOR "Joe Bloggs"
!build_title

H1: Introduction

After extensive market research, I believe there is an excellent opportunity for us to develop software for the galaxy construction industry. Potential customers include:

* NASA
* European Community
* China
* Japan.

Note: The proposed name of the software package to be developed is [[DOC_NAME]]. If you want to suggest a better name, send email to {{EMAIL:joe@bloggs.com}}.

H2: Software Requirements

The key requirements are:

^ support for the design and simulation of galaxies containing up to:
- 1000 large planets, or
- 5000 small planets
+ the package needs to be easy to use
+ the package needs to be well documented.

H2: Project Team

Exploding galaxies will be very bad for business, so we need the best team possible for this project:

!block table
Person          Role
Mary Jones      Project Manager
Hans Blass      Architect
Bill Smith      Software Engineer
!endblock

Note: This document (called mydoc.sdf) is provided in the doc/paper directory of the SDF distribution.
Comments begin with a # character as the first non-whitespace character on a line.

Macros are embedded commands which begin with a ! as the first non-whitespace character on a line. The define macro is used to set variables. The value of a variable can be embedded in paragraph text by using the [[...]] syntax.

The DOC_NAME and DOC_TYPE variables are used by the build_title macro which creates:
- a cover page (or two) for paper-based outputs
- a title header for online outputs.

Paragraphs can be tagged in different ways. For the vast majority of SDF documents, the only tags used are:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1:</td>
<td>level 1 heading</td>
</tr>
<tr>
<td>H2:</td>
<td>level 2 heading</td>
</tr>
<tr>
<td>*</td>
<td>item in level 1 bulleted list</td>
</tr>
<tr>
<td>-</td>
<td>item in level 2 bulleted list</td>
</tr>
<tr>
<td>^</td>
<td>first item in level 1 ordered list</td>
</tr>
<tr>
<td>+</td>
<td>next item in level 1 ordered list</td>
</tr>
<tr>
<td>&gt;</td>
<td>fixed-width, verbatim text</td>
</tr>
</tbody>
</table>

Phrases can also be tagged in several ways. Any phrase can be tagged using the syntax:

`{{XYZ:...}}`

where XYZ is the tag. For single, uppercase character tags like I (Italics) and B (Bold), POD-style syntax is also supported:

`<...>`

where X is the tag.
Getting Started
A simple example

Tables can be specified using the `table` filter, typically in combination with the `block` and `endblock` macros. The first row is the headings. Remaining rows are data.
Generating outputs

Overview

The `sdf` command is used to convert SDF to other formats. The general syntax is:

```
sdf [options] file ...
```

If an extension is not given (and a file is not found with that name), an extension of `sdf` is assumed.

Generating HTML

To convert `mydoc.sdf` to HTML, the command is:

```
sdf -2html mydoc
```

This creates a file called `mydoc.html`.

Generating a set of HTML topics

To convert `mydoc.sdf` to a set of HTML topics, the command is:

```
sdf -2topics mydoc
```

This creates the following files:

- `mydoc.html` - the table of contents
- `mydoc_1.html` - the first topic
- `mydoc_2.html` - the second topic.

Generating PostScript on Unix

To convert `mydoc.sdf` to PostScript on Unix, the command is:

```
sdf -2ps mydoc
```

This creates a file called `mydoc.ps` by:

1. Generating a MIF (Maker Interchange Format) file called `mydoc.out`.
2. Using the `fmbatch` program to convert this file to `mydoc.out.ps`.
3. Renaming `mydoc.out.ps` to `mydoc.ps`.
4. Deleting `mydoc.out`.

Generating PostScript on Windows

To convert `mydoc.sdf` to PostScript on Windows, the command is:

```
sdf -2mif mydoc
```

This creates a file called `mydoc.mif` which can be opened in FrameMaker and printed to a PostScript file.
Generating PDF

To generate a PDF file, the steps are:

1. Generate a PostScript document (see above)
2. Open the PostScript document using Adobe Distiller.
3. Generate the PDF file.

Note: The PostScript document generated by SDF includes the information needed by the PDF distiller to generate a table of contents.

Generating plain text

To convert mydoc.sdf to plain text, the command is:

```
sdf -2txt mydoc
```

This creates a file called mydoc.txt.

Generating Windows help

To convert mydoc.sdf to Windows help, the command is:

```
sdf -2hlp mydoc
```

This creates the following files:

- `mydoc.hpj` - a help project file
- `mydoc.rtf` - an RTF document with embedded help codes

These need to be compiled using a help compiler (e.g. hcp.exe). For example:

```
hcp mydoc
```

This creates a file called `mydoc.hlp` (provided no errors are encountered).
Document styles

Changing the style

The general style (or type) of a document can be changed by initialising the OPT_STYLE variable on the top line of a document. For example:

```
!init OPT_STYLE="manual"
```

Alternatively, sdf’s -s option can be used. For example:

```
sdf -2ps -smanual mydoc
```

**Note:** The -s option overrides the *init* macro setting.

Available styles

The available document styles, grouped by *style category*, are:

<table>
<thead>
<tr>
<th>Style</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General:</strong></td>
<td></td>
</tr>
<tr>
<td>document</td>
<td>a normal document</td>
</tr>
<tr>
<td>manual</td>
<td>a manual</td>
</tr>
<tr>
<td>paper</td>
<td>a technical paper</td>
</tr>
<tr>
<td><strong>Administration:</strong></td>
<td></td>
</tr>
<tr>
<td>admin</td>
<td>generic administration document</td>
</tr>
<tr>
<td>fax</td>
<td>a fascimile</td>
</tr>
<tr>
<td>memo</td>
<td>a memorandum</td>
</tr>
<tr>
<td>newsltr</td>
<td>a newsletter</td>
</tr>
<tr>
<td>minutes</td>
<td>minutes of a meeting</td>
</tr>
<tr>
<td><strong>Miscellaneous:</strong></td>
<td></td>
</tr>
<tr>
<td>listing</td>
<td>a source code listing</td>
</tr>
</tbody>
</table>

Building a title

Each style category has a different way of building a title section:

- *general* styles use the `build_title` macro
- *administration* styles use the `title` filter
- *miscellaneous* styles do not have a title.
Creating new styles

It is relatively simple to create new styles by inheriting details from an existing one.

Refer to the *SDF Guru Guide* for details.
Building a title

Each document style category has a different way of building a title section:

- **administration** styles use the `title` filter
- **general** styles use the `build_title` macro
- **miscellaneous** styles do not have a title.

Refer to *Document styles* for a summary of the styles available and the category of each style.

### The title filter

The `title` filter generates a title block for administration-style documents like memos and faxes. For example:

```
!block title
Name        Value  
Date:       
[[DATE:DOC_MODIFIED]]
To:         Joe Bloggs, Sue Brown, Maree Jones
Copy:       David Smith
From:       Neil Armstrong
Subject:    Solar System Information
Ref. No:    XY.002/GUI/96
!endblock
```

Refer to the documentation on the `title` filter within the *SDF Reference* for further details.

### The build_title macro

The `build_title` macro tunes the layout and content of the title section for each target format. Typically:

- paper-based documents have all the title information formatted on a page (or two)
- online documents only have the essential title information formatted in a few lines.

By convention, parameters are passed to `build_title` via predefined variable names. For example:

```
!define DOC_NAME           "GalaxyBuilder"
!define DOC_TYPE           "User Guide"
!define DOC_AUTHOR         "Joe Bloggs"
!build_title
```

### Variables used by build_title

The following variables are commonly defined before `build_title` is called:
Refer to the documentation on the build_title macro within the SDF Reference for further details.
Chapter 2. Language overview

Basic concepts

The basic concepts within SDF documents are:

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>paragraph</td>
<td>one or more lines of text</td>
</tr>
<tr>
<td>phrase</td>
<td>a section of text within a paragraph</td>
</tr>
<tr>
<td>style</td>
<td>the type of a document, paragraph, phrase or table (e.g. H1)</td>
</tr>
<tr>
<td>macro</td>
<td>a command embedded in a document (e.g. !define)</td>
</tr>
<tr>
<td>variable</td>
<td>a named value (e.g. DOC_NAME)</td>
</tr>
<tr>
<td>filter</td>
<td>a rule to use when processing certain sections of text (e.g. table)</td>
</tr>
<tr>
<td>attribute</td>
<td>a named parameter of a paragraph, phrase or filter (e.g. jump)</td>
</tr>
<tr>
<td>expression</td>
<td>a literal or expression to evaluate (e.g. &quot;Ian Clatworthy&quot;).</td>
</tr>
</tbody>
</table>

Further details about these are given below.
Paragraphs

Introduction

Paragraphs have the following format:

```
line1
:::
lineN
```

Leading and trailing whitespace on lines is generally ignored. Paragraphs are separated by blank lines or:

- *comment lines* - first non-whitespace character is ";"
- *macros* - first non-whitespace character is "!".

For normal paragraphs, simply specify the text on one or more lines. For example:

```
I like products which are simple to use and
do what I expect. We should encourage engineers
to design more products with these qualities.
```

Paragraph styles

A paragraph can be given a style using the following syntax:

```
style::"line1
:::
lineN
```

Tagged paragraphs do not need to be separated by blank lines. *style* is a sequence of one or more of the following characters:

- letters
- digits
- underscore character (_).

Whitespace is not permitted before the "::" but is generally ignored immediately after it. The style is optional and defaults to N (for Normal). The text is also optional.

Commonly used styles

Commonly used paragraph styles are given below.
For example:

Note: Life is too short to drink bad wine.

The result is:

Note: Life is too short to drink bad wine.

Special styles

For certain styles, the following syntax is also supported:

```
special_style line1
   ...
lineN
```

The special styles available are:

<table>
<thead>
<tr>
<th>Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>. .. ......</td>
<td>paragraph or plain list item at level 1-6</td>
</tr>
<tr>
<td>* .. ******</td>
<td>unordered list at level 1-6</td>
</tr>
<tr>
<td>- .. -----</td>
<td>unordered list at level 2-6</td>
</tr>
<tr>
<td>^ .. ^^^^^^</td>
<td>first entry in an ordered list at level 1-6</td>
</tr>
<tr>
<td>+ .. ++++++</td>
<td>next entry in an ordered list at level 1-6</td>
</tr>
</tbody>
</table>

For example:

* fruits:
  - peach
  - banana
* vegetables:
  - potato
  - carrots.
The result is:

- fruits:
  - peach
  - banana
- vegetables:
  - potato
  - carrots.
Phrases

Introduction

A *phrase* is a section of text within a paragraph enclosed in the symbols {{ and }}. Like paragraphs, phrases are optionally tagged with a style. Some examples are given below.

<table>
<thead>
<tr>
<th>SDF</th>
<th>Typical result</th>
</tr>
</thead>
<tbody>
<tr>
<td>{{Hello world}}</td>
<td>Hello world</td>
</tr>
<tr>
<td>{{1:Hello world}}</td>
<td>Hello world</td>
</tr>
<tr>
<td>{{2:Hello world}}</td>
<td>Hello world</td>
</tr>
<tr>
<td>{{B:Hello world}}</td>
<td>Hello world</td>
</tr>
<tr>
<td>{{I:Hello world}}</td>
<td>Hello world</td>
</tr>
<tr>
<td>{{SECT:Tables}}</td>
<td>Tables</td>
</tr>
<tr>
<td>{{FILE:autoexec.bat}}</td>
<td>autoexec.bat</td>
</tr>
</tbody>
</table>

Commonly used styles

Some commonly used phrase styles are given below.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Emphasis:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1st level emphasis (default)</td>
<td><em>emphasis 1</em></td>
</tr>
<tr>
<td>2</td>
<td>2nd level emphasis</td>
<td><em>emphasis 2</em></td>
</tr>
<tr>
<td>3</td>
<td>3rd level emphasis</td>
<td><em>emphasis 3</em></td>
</tr>
<tr>
<td>ST</td>
<td>strong emphasis</td>
<td><strong>strong emphasis</strong></td>
</tr>
<tr>
<td><strong>Formatting:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>normal</td>
<td>some normal text</td>
</tr>
<tr>
<td>I</td>
<td>italic</td>
<td><em>some italic text</em></td>
</tr>
<tr>
<td>B</td>
<td>bold</td>
<td><em>some bold text</em></td>
</tr>
<tr>
<td>U</td>
<td>underline</td>
<td><em>some underline text</em></td>
</tr>
<tr>
<td>EX</td>
<td>example</td>
<td><em>some example text</em></td>
</tr>
<tr>
<td><strong>Types:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMAIL</td>
<td>email address</td>
<td><a href="mailto:ianc@mincom.com">ianc@mincom.com</a></td>
</tr>
<tr>
<td>FILE</td>
<td>Filename</td>
<td>myfile.sdf</td>
</tr>
<tr>
<td>SECT</td>
<td>Section</td>
<td><em>Paragraphs</em></td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
<td><a href="http://www.mincom.com">http://www.mincom.com</a></td>
</tr>
</tbody>
</table>
Language overview

Phrases

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOC</td>
<td>document title</td>
<td><em>SDF User Guide</em></td>
</tr>
<tr>
<td>REF</td>
<td>reference (document code)</td>
<td>MTR-SDF-0002</td>
</tr>
<tr>
<td>ORG</td>
<td>organisation</td>
<td>Mincom</td>
</tr>
<tr>
<td>PRD</td>
<td>product</td>
<td>MIMS</td>
</tr>
</tbody>
</table>

**Note:** When emphasising text, it is generally better to use an emphasis style rather than a formatting style as the best way of doing so depends on the output format.

**Type vs class styles**

A *type* (e.g. EMAIL) simply marks a phrase as a logical entity. Rules may be defined for processing (e.g. generating hypertext) for these types.

A *class* (e.g. DOC) is a special kind of type where the entity must be a member of a predefined set. Rules can also be defined for processing classes, although hypertext jumps are often defined for each entity in the tables which define the known entities. Refer to *Object Management*, later.
Macros

Introduction

A macro is a command which can be embedded within SDF. Macros begin with an exclamation mark (!) as the first non-whitespace character on a line. Some examples are:

```
!use "delphi"
!define DOC_AUTHOR "Joe Bloggs"
!build_title
```

Commonly used macros

Some commonly used macros are given below.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>init variables</td>
<td>initialise variables</td>
</tr>
<tr>
<td>define variable [expression]</td>
<td>set a variable's value</td>
</tr>
<tr>
<td>build_title</td>
<td>build a title page</td>
</tr>
<tr>
<td>block filter</td>
<td>begin a block of text</td>
</tr>
<tr>
<td>endblock</td>
<td>end a block of text</td>
</tr>
<tr>
<td>include file[:filter]</td>
<td>include another file</td>
</tr>
<tr>
<td>import file[:parameters]</td>
<td>import a figure</td>
</tr>
</tbody>
</table>

A complete list of the supported macros and their parameters is given in the SDF Reference manual.

Creating new macros

The easiest way to create a new macro is to use the macro and endmacro macros. For example:

```
!macro MY_MACRO
This text will appear whenever the author calls !MY_MACRO.
!endmacro
```

Macros can also be implemented by a Perl subroutine which generates the SDF text to be inserted when the macro is called. See the SDF Guru Guide for further details.
Variables

Introduction

A variable is a named value. Document-wide settings are controlled in SDF using variables. Likewise, authors can define and access their own variables. In either case, the value of a variable can be referenced in a paragraph by delimiting it with the special symbols [[ and ]].

For example:

```
!define MY_EMAIL 'abc@xyz.com'
My electronic mail address is [[MY_EMAIL]].
```

The result is:

My electronic mail address is abc@xyz.com.

Commonly used system variables

Some commonly used system variables are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Options:</td>
<td></td>
</tr>
<tr>
<td>OPT_LOOK</td>
<td>the overall look (e.g. simple, fancy)</td>
</tr>
<tr>
<td>OPT_STYLE</td>
<td>the general type (e.g. memo, manual)</td>
</tr>
<tr>
<td>Titles:</td>
<td></td>
</tr>
<tr>
<td>DOC_NAME</td>
<td>the title, excluding the type (e.g. SDF)</td>
</tr>
<tr>
<td>DOC_TYPE</td>
<td>the title type (e.g. User Guide)</td>
</tr>
<tr>
<td>DOC_AUTHOR</td>
<td>the author</td>
</tr>
<tr>
<td>DOC_TOC</td>
<td>the number of heading levels in the table of contents</td>
</tr>
</tbody>
</table>

Setting variables

Variables can be set via:

- the `define` macro
- the `default` macro
- the `init` macro
- `sdf`’s command line.

To undefine a variable, use the `undef` macro.
The Define macro

Variables are usually set in SDF by using the `define` macro. The format is:

```
!define symbol [expression]
```

Examples are:

```
!define LONG
!define VERSION "1.002"
```

`symbol` is a sequence of letters, digits and underscore. If `expression` is omitted, the default value is 1.

The Default macro

It is often a good idea to only define a variable if it does not already have a value. To do this, use the `default` macro. The format is:

```
!default symbol [expression]
```

For example:

```
!default VERSION "1.002"
```

`symbol` is a sequence of letters, digits and underscore. If `expression` is omitted, the default value is 1.

The Init macro

Generally speaking, the first thing `sdf` does is to load the standard library (`stdlib.sdm`). However, if the first line of an SDF document is the `init` macro, the nominated variables are initialised before the standard library is loaded. This allows an SDF document to initialise variables used within the standard library. The format of the init macro is:

```
!init name1["="expression1"];" name2["="expression2]
```

For example:

```
!init OPT_STYLE="memo"
```

**Note:** Certain variables cannot be set after the standard library is loaded. These variables can only be set via `sdf`'s command line or via the `init` macro.

Using the command line

It is occasionally useful to set variables via `sdf`'s `-D` option. Multiple definitions can be separated by commas. For example:

```
sdf -DLONG,VERSION="1.002" ...
```
Alternatively, the option can be repeated. For example:

```
sdf -DLONG -DVERSION="1.002" ...
```

In either case, if a value is not specified for a variable, 1 is assumed.

**Note:** Variables set on the command line override the values set by the `init` macro.
Filters

Introduction

A filter controls how a block of text is interpreted. The text is usually delimited by \texttt{block} and \texttt{endblock} macros.

For example, tables are usually defined via the \texttt{table} filter:

\begin{verbatim}
!block table
Option    Description
-h        display help
-o        specify the output extension
!endblock
\end{verbatim}

The result is:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>display help</td>
</tr>
<tr>
<td>-o</td>
<td>specify the output extension</td>
</tr>
</tbody>
</table>

Commonly used filters

Some of the commonly used filters are:

<table>
<thead>
<tr>
<th>Filter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{table}</td>
<td>the lines are a table in SDF's \texttt{TBL} format</td>
</tr>
<tr>
<td>\texttt{example}</td>
<td>the lines are example paragraphs</td>
</tr>
<tr>
<td>\texttt{title}</td>
<td>used to build a title block for memos, faxes, etc.</td>
</tr>
<tr>
<td>\texttt{topics}</td>
<td>include files as sub-topics</td>
</tr>
<tr>
<td>\texttt{appendix}</td>
<td>replace H1 styles with A1, etc.</td>
</tr>
<tr>
<td>\texttt{plain}</td>
<td>replace H1 styles with P1, etc.</td>
</tr>
</tbody>
</table>

A complete list of the supported filters and their parameters is given in the \textit{SDF Reference} manual.

Other macros supporting filters

Other macros also support filters. These include:

- \texttt{include} - include text from another file
- \texttt{execute} - include the output of a system command.

For example, the following line includes another SDF file and formats it as an appendix:
Language overview
Filters

!include "tips.sdf"; appendix

Note: The *appendix* and *plain* filters enable authors to construct topics without needing to worry about how those topics will be used, e.g. a topic may be a chapter in one document and an appendix in another!
Attributes

Introduction

Attributes can be used to specify custom formatting, hypertext targets and jumps, indexing information, etc. For example:

Note[label='Important: ';size="16pt"]
Life is too short to drink bad wine.

The result is:

**Important:** Life is too short to drink bad wine.

Syntax

Paragraphs and phrases can be given attributes via the syntax below:

```
style"["attributes"]" text
```

where the syntax of *attributes* is:

```
name1[="expression1"];" name2[="expression2"] ...
```

If an attribute name is given without a value, the default value is 1. Like style names, attribute names are:

- sequences of letters, digits and "_"
- case-sensitive.

Commonly used attributes

Some commonly used attributes are given below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Paragraphs:</strong></td>
<td></td>
</tr>
<tr>
<td>label</td>
<td>prefix label (e.g. Note:)</td>
</tr>
<tr>
<td>notoc</td>
<td>take this heading out of the table of contents</td>
</tr>
<tr>
<td><strong>Phrases:</strong></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>hypertext target tag</td>
</tr>
<tr>
<td>jump</td>
<td>URL (Uniform Resource Locator) to jump to</td>
</tr>
</tbody>
</table>
Expressions

Introduction
Expressions are used in several places within SDF including:

- attribute values
- macro argument values
- filter parameter values
- evaluation within paragraph text.

The value of an expression can be placed into a paragraph by delimiting it with the special symbols [[ and ]]. For example:

   The current file is [[DOC_FILE]].

Types of expressions
An expression is either:

- an SDF variable name (e.g. DOC_AUTHOR)
- a Perl expression.

If an expression is a single word, it is the value of a variable. Otherwise, it is evaluated as a Perl expression. A Perl expression is typically one of the following:

- a numeric literal (e.g. 213)
- a string literal (e.g. "User Guide")
- a subroutine call (e.g. length("User Guide")).

Accessing variables within Perl expressions
Within a Perl expression, SDF variables are available via the var associative array. The syntax used by Perl for accessing a value within an associative array is:

   "$" array_name "{" string_index "}""}

Some examples are:

1. get the length of the DOC_AUTHOR variable:
   
   \[\text{length}\left(\text{var}'\text{DOC_AUTHOR}'\right)\]

2. concatenate the document name and type:
   
   \[\text{var}'\text{DOC_NAME}' \text{ var}'\text{DOC_TYPE}'\]
Chapter 3. Paragraphs

Headsings

Chapter Headings
Headings are specified using the tags H1 .. H6:

• H1 is a chapter heading
• H2 .. H6 are sub headings

The actual presentation of H-style headings (e.g. numbered or unnumbered) is decided by the target format and document style.

Other Headings
The following heading tags are also supported:

• A1 .. A6 - headings in Appendices
• P1 .. P6 - headings in plain sections (e.g. Glossary)

P-style headings are always unnumbered so are commonly used in memos.

In normal documents, it is often unnecessary to use A-style or P-style tags. The preferred approach is to:

• store each section in a file using H-style headings
• convert the heading style using the appendix or plain filter as the file is being imported.

For example:

!include "budget.sdf"; appendix

Using this approach, sections can easily be reused in different ways in different documents.

Generating a Table of Contents
A table of contents can be generated from the heading paragraphs. All types of heading tags (i.e. H, A and P) are included. The depth of the table of contents is specified by the DOC_TOC variable. For example, to generate a table of contents including headings up to and including level 4:

doccvt -o -DDOC_TOC=4 mydoc.sdf
**Note:** For normal documents, the *build_title* macro defaults this variable to 3.

Individual headings can be excluded from the table of contents by specifying the *notoc* attribute. For example:

```sdf
H3[notoc] Yet Another Heading
```

See *Attributes* for further details on attribute syntax.

**Controlling Page Breaks**

Generally speaking, page breaks are controlled by rules associated with heading tags. For example:

- for documents, memos and faxes, each level 1 heading starts a new page
- for manuals, each level 1 heading starts a new chapter and each level 2 heading typically starts a new page.

The *top* paragraph attribute can be used to position a paragraph at the top of the next page. For example:

```sdf
H3[top] Abc Module
```

Page breaks can also be explicitly specified using the *PB* tag. For example:

```sdf
text on one page.
PB:
H3: Another Section
```

**Note:** SDF is designed to be a *logical markup language*. While it is sometimes necessary to explicitly specify page breaks in order to get the layout you want, doing so can reduce the portability of your document to different styles, page sizes and target formats.
Example Paragraphs

Simple Examples

Example paragraphs are displayed in a fixed-width font (e.g. Courier). Unlike most other paragraphs, spaces and tabs immediately after the paragraph tag are not ignored in example paragraphs. The tags used are:

- E - fixed-width text
- V - verbatim, fixed-width text (embedded symbols are ignored).

For example:

```
E:#{ {Greeting}} outputs a friendly greeting.
E:sub Greeting {
E:  print "Hello world\n";
E:}

The result is:

# Greeting outputs a friendly greeting.
sub Greeting {
  print "Hello world\n";
}
```

Shorthand Notation

As verbatim, fixed-width text is very common in software documentation, "->" is provided as a shorthand for "V:”. For example:

```
>#{ {Greeting}} outputs a friendly greeting.
>sub Greeting {
  >print "Hello world\n";
  >}

The result is:

# {{Greeting}} outputs a friendly greeting.
sub Greeting {
  print "Hello world\n";
}
```

Large Examples

Large examples are usually defined using the example or verbatim filters like this:

```
!block example
# {{Greeting}} outputs a friendly greeting.
sub Greeting {
  print "Hello world\n";
}
!endblock
```

Wide Examples

If an example block has unwanted line breaks, the wide parameter should be specified. Wide examples use a wider text area (and a smaller font, if necessary) so that 80 characters can fit onto each line. For example:
Example Paragraphs

Example Paragraphs

!block example; wide
sub SaveTheWorld {                  # This routine provides world peace
  local($param1, param2) = @_;    # Input parameters
    # Do the work
    # ...
}
!endblock

The result is:

sub SaveTheWorld {                  # This routine provides world peace
  local($param1, param2) = @_;    # Input parameters
    # Do the work
    # ...
}

Formatting Source Code

Blocks of source code can be nicely formatted via the lang parameter to the example filter. For example:

!block example; lang='Perl'
sub hello {                           # Output a nice message
  local($planet) = @_;               # Output a nice message
    print "hello $planet!\n";
}
!endblock

The result is:

sub hello {                           # Output a nice message
  local($planet) = @_;               # Output a nice message
    print "hello $planet!\n";
}

For convenience, if a programming language is used as a filter (and a filter of that name doesn't exist), SDF implicitly calls the example filter for you with the relevant lang parameter.

There is built-in support for numerous languages including Perl, C, C++, Java, Delphi, CORBA IDL and shell. New language definitions can be easily added (vgrind definitions are used).

File Pretty Printing

Pretty printing of source code files is directly supported by sdf's -P option. For example:

sdf -2ps -Psh myscript
sdf -2ps -P myapp.c
sdf -2ps -P -n5 mylib.pl
The language to use can be specified as a parameter. The default language is derived from the extension of the file. The -n option adds line numbers at the frequency given. The default frequency is 1. i.e. every line.
Lists

Overview

SDF supports 3 types of lists:

- *unordered* - items are bulleted
- *plain* - items are not bulleted
- *ordered* - items are labelled with numbers or letters.

As lists are so common, they are supported by a special syntax:

```
special_tag text
```

where `special_tag` is a sequence of 1 to 6 special characters. Spaces or tabs between the tag and text are ignored.

Unordered Lists

To define an unordered list, use the `*` character. For example:

```
* peach
* banana
* mango.
```

The result is:

- peach
- banana
- mango.

Plain Lists

To define a plain list, use the `.` character. For example:

```
. peach
. banana
. mango.
```

The result is:

- peach
- banana
- mango.

Ordered Lists

To define an ordered list, use the `^` character for the first item and the `+` character for remaining items. For example:
^ peach  
+ banana  
+ mango.

The result is:
1. peach  
2. banana  
3. mango.

List Paragraphs
To nest a normal paragraph inside a list, use the `.` tag. For example:

```
* peach
. This is one of my favorites.
```

The result is:

- peach
  This is one of my favorites.

Nested Lists
Lists can be nested up to 6 levels. The nesting level is taken from the number of special characters in the tag. For example:

```
** a bulleted list at level 2
^^^ an ordered list at level 3
```

To nest normal paragraphs inside these lists, use plain list items. For example:

```
** a bulleted list at level 2
.. A normal paragraph at level 2.
```

The `-` Special Tag
As a convenience, the `-` character can be used to specify a bulleted list at level 2. For example:

```
* inputs:
  - programmers
  - pizza
* outputs:
  - software
  - empty pizza boxes.
```

The result is:

- inputs:
  - programmers
  - pizza
- outputs:
– software
– empty pizza boxes.

Likewise, the tags '---' to '-----' are equivalent to the tags '***' to '******'.
Notes

Short Notes

The paragraph tag for adding a single paragraph note is `Note`. For example:

```plaintext
Note: Life is too short to waste time being unhappy.
```

The result is:

```
Note: Life is too short to waste time being unhappy.
```

Changing the Label

You can change the label for a note by using the `label` paragraph attribute. For example:

```plaintext
Note[label='Important: '] Life is too short to drink bad wine.
```

The result is:

```
Important: Life is too short to drink bad wine.
```

Long Notes

If the note requires more than one paragraph, use the `note` filter. For example:

```plaintext
!block note
Life is too short to:

* waste time being unhappy
* worry about yesterday.
!endblock
```

The result is:

```
Note: Life is too short to:
• waste time being unhappy
• worry about yesterday.
```
The NB/NE Tags

Before the `note` filter existed, long notes were created by using the NB (Note Begin) and NE (Note End) tags. For example:

NB: Life is too short to:

* waste time being unhappy
* worry about yesterday.

NE:

These paragraph styles are still supported, although the `note` filter is now the recommended way of specifying long notes.
Chapter 4. Figures

Overview

Features

SDF supports a range of features related to figures:

- figures can be imported in a variety of formats including:
  - EPS - Encapsulated PostScript
  - WMF - Windows MetaFile
  - MIF - Maker Interchange Format
  - GIF - Graphics Interchange Format
  - JPEG - JPEG File Interchange Format
  - PNG - Portable Network Graphic Format
  - BMP - Windows Bitmap Format

- logos can be easily added to title pages and online headers
- figure positioning can be controlled
- text can be wrapped around figures.

Importing Figures Into SDF Documents

The \texttt{import} macro is usually used to import a figure. The syntax is:

\begin{verbatim}
!import "filename" [; parameters]
\end{verbatim}

For example:

\begin{verbatim}
!import "sdfarch"
\end{verbatim}

Importing a Figure Into Paragraph Text

To import a figure into the middle of a paragraph, the \texttt{IMPORT} phrase style is used. The syntax is:

\begin{verbatim}
{{IMPORT:filename}}, or
{{IMPORT[parameters]filename}}
\end{verbatim}

For example:

Please visit our {{IMPORT:new}} web site!

Figure Extensions

It is generally best \textit{not} to specify an extension for the filename containing the figure, as SDF will then use the best format it can find. The search rules are:
So a GIF file is all that is necessary if you are generating PostScript and HTML. However, if SDF finds an EPS file when it is generating PostScript, it will use that instead.

Adding Logos

The default implementations of the build_title macro, title filter and HTML header macros provide an easy way to add a company logo: by defining variables. The variables are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOC_LOGO</td>
<td>the default logo</td>
</tr>
<tr>
<td>DOC_TITLE_LOGO</td>
<td>used in the title of a PostScript document</td>
</tr>
<tr>
<td>DOC_ADMIN_LOGO</td>
<td>used in the title of a PostScript admin. document</td>
</tr>
<tr>
<td>DOC_HTML_LOGO</td>
<td>used in the title of a HTML document</td>
</tr>
<tr>
<td>DOC_TOPIC_LOGO</td>
<td>used in the header of a HTML topic</td>
</tr>
<tr>
<td>DOC_MANUAL_LOGO</td>
<td>used on the front cover of a PostScript manual</td>
</tr>
<tr>
<td>DOC_COMPONENT_LOGO</td>
<td>used on the cover of each chapter in a PostScript manual</td>
</tr>
</tbody>
</table>

In each case, the value of the variable is the filename containing the figure.

If a given logo variable is not set, DOC_LOGO is used if it is set. Therefore, a company logo can be added to most SDF documents by simply defining DOC_LOGO is a configuration module (or library) which is loaded into every SDF document via the use (or inherit) macro.
Creating Figures

Types of Figure Formats

Figure formats are typically either:

- **vector** based, i.e. a set of drawing instructions
- **bitmap** based, i.e. a set of pixels.

For diagrams which look good when printed, a vector format (e.g. EPS) is required. For window snapshots, a bitmap format (e.g. GIF) is sufficient.

Generating Bitmap-based Figures

With the current popularity of the World Wide Web, most diagramming and image processing tools can now export JPEG, PNG and/or GIF formats. Therefore, if HTML is the only output format you care about for a given document, getting a figure into it is easy.

Occasionally, GIF format is also sufficient for paper-based documents. However, a GIF figure which displays fine in a browser may or may not display correctly in a PostScript document! The reason is that FrameMaker 5.x has problems importing:

- certain GIFs in 89a format
- GIFs with interlacing
- GIFs with transparent backgrounds.

Therefore, if you want to import a GIF file into a PostScript document, you will probably need to stick with GIF 87 format and avoid anything fancy.

Generating Vector-based Figures

For good quality diagrams, you will need to export your figure from your drawing application into a vector-based format that FrameMaker can successfully import.

The following information may be useful in selecting a vector-based format:

- 99% of Windows tools have a good WMF export filter, but EPS export is rarely supported and often broken (Visio 3.0 and Adobe Illustrator are known exceptions)
- FrameMaker 5.x on Solaris 2.x has a good EPS import filter, but the WMF import filter seems buggy
Figures
Creating Figures

- FrameMaker 5.x on Windows 95/NT has a good WMF import filter.

So, if you are using FrameMaker on Windows 95/NT, WMF is the vector format to use and generating it from a Windows-based tool should be easy. However, if you are using FrameMaker on Solaris, EPS is the vector format to use, but generating it from a Windows-based tool might be tricky.

---

Generating EPS From Windows Applications

The following approach can often be used to generate an EPS file from a Windows program:

1. Print PostScript to a file, typically via a PostScript printer configured to use the FILE: port.
2. Run Ghostscript's ps2epsi program on the file.

Depending on the combination of application and printer driver, you may need to hand edit the ps file before you run ps2epsi. See the SDF Frequently Asked Questions for examples.

---

Generating FrameMaker Figures

FrameMaker can be used to produce figures which SDF can import when it is generating PostScript. To do this, the steps are:

1. Create an anchored frame.
2. Draw the figure within it using FrameMaker's drawing tools.
3. Save the file into MIF format.

The main disadvantage with this approach is that FrameMaker 5.x does not provide an easy way to convert the figure to other formats (e.g. GIF).
Adding Figure Titles

Adding a Title to a Figure
To add a title, use the `title` attribute of the `import` macro. For example:

```plaintext
!import "growth"; title="Predicted Growth"
```

Adding a Title to a Psuedo Figure
Occasionally, it is useful to label a source code example or ASCII graphic with a title. To do this, use the `FT` (Figure Title) paragraph style. For example:

```plaintext
FT: Source code for main.c
!include "main.c"; example; lang='C'
```

Generating a List of Figures
A list of figures can be generated from the figure titles. To do this, set the `DOC_LOF` variable. For example:

```plaintext
# Build the title
!define DOC_NAME "World Peace in 60 Days"
!define DOC_LOF
!build_title

This will generate a list of figures after the table of contents.

Changing the List of Figures Title
The default title for the list of figures is "List of Figures". To change this, set the `DOC_LOF_TITLE`. For example:

```plaintext
!define DOC_LOF_TITLE "Figures"
```
Positioning a Figure

The alignment of a figure can be specified using the `align` attribute of the `import` macro. The supported values are:

<table>
<thead>
<tr>
<th>Align</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Center</td>
<td>centered within the region (the default)</td>
</tr>
<tr>
<td>Left</td>
<td>left side of region</td>
</tr>
<tr>
<td>Right</td>
<td>right side of region</td>
</tr>
<tr>
<td>Inner</td>
<td>inner side of region for double sided documents (otherwise Left)</td>
</tr>
<tr>
<td>Outer</td>
<td>outer side of region for double sided documents (otherwise Right)</td>
</tr>
</tbody>
</table>

For example:

```
!import "my_graph"; align=Left
```

Wide Figures

By default, figures are aligned within the main text area. To align a figure within the full text area (i.e. including the side-head), use the `wide` attribute. For example:

```
!import "my_graph"; align=Left; wide
```

Figures Within a List

To align a figure with a given list indent, use the `listitem` attribute. For example:

```
* The graphs are impressive:
  - savings per year
    !import "my_graph"; listitem=2
```

Vertical Positioning

The vertical position of a figure can be specified via the `position` attribute. The supported values are:

<table>
<thead>
<tr>
<th>Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Below</td>
<td>below the current paragraph (the default)</td>
</tr>
<tr>
<td>Top</td>
<td>top of the current column/page</td>
</tr>
<tr>
<td>Bottom</td>
<td>bottom of the current column/page</td>
</tr>
</tbody>
</table>

For example:
Wrapping Text Around a Figure

To wrap text around a figure, set the `wrap_text` attribute. For example:

```
!import "my_graph"; wrap_text
```

**Note:** When generating HTML, text can be wrapped around a figure but unfortunately, the rules are quite different:

1. The `wrap_text` attribute has no affect.
2. Text is implicitly wrapped if the `align` attribute is set to Left or Right.
3. The wrapping must be explicitly disabled using the `clear` macro.
Figures
Positioning a Figure
Chapter 5. Tables

Overview

Features
SDF supports a range of features related to tables including:

• tables are defined in a simple format which is:
  – easy to write and read
  – upwardly-compatible with the CSV (comma-separated value) format exported by most databases and spreadsheets
  – upwardly-compatible with the fixed-width format generated by most reporting tools.

• column widths can be dynamically sized or explicitly set

• heading, footing and group rows can be specified

• table alignment and positioning can be controlled

• cells support custom alignment, shading, ruling and colours.

In fact, SDF can do lots of very clever things with tables including spreadsheet-like calculations on cells and ranges of cells!

Creating a Table
Tables are usually defined via the \texttt{table} filter. For example:

\begin{verbatim}
!block table
Fruit   Comments
Tropical:
mango  a fruit-bat favorite
banana great for smoothies
Other:
Tomato often matched with basil
!endblock
\end{verbatim}
or:

\begin{verbatim}
!include "fruits.tbl"; table
\end{verbatim}

TBL Format
The \texttt{table} filter expects a table definition in a simple text format (called \texttt{TBL}) where:

1. the first (non-comment) line is the header record
2. remaining lines are data records.
The header record specifies the field names and the layout. Fields can be placed in fixed-width columns or delimited by a special character. Fixed-width format (as shown above) is usually used.

**Note:** The width of columns in the "input" has no bearing on the width of columns in the output. To change the output width, refer to *Sizing Columns*.

See the *SDF Reference* for further details on *TBL* format.
Table Styles

The style Attribute

The look of a table is controlled by the *style* attribute. For example:

```html
!block table; style="rows"
Name   Phone
Bill   123 4567
Joe    789 1234
Mary   584 3333
!endblock
```

The result is:

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill</td>
<td>123 4567</td>
</tr>
<tr>
<td>Joe</td>
<td>789 1234</td>
</tr>
<tr>
<td>Mary</td>
<td>584 3333</td>
</tr>
</tbody>
</table>

The other styles available are shown below.

columns

```html
<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill</td>
<td>123 4567</td>
</tr>
<tr>
<td>Joe</td>
<td>789 1234</td>
</tr>
<tr>
<td>Mary</td>
<td>584 3333</td>
</tr>
</tbody>
</table>
```

plain

```html
<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill</td>
<td>123 4567</td>
</tr>
<tr>
<td>Joe</td>
<td>789 1234</td>
</tr>
<tr>
<td>Mary</td>
<td>584 3333</td>
</tr>
</tbody>
</table>
```

grid

```html
<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill</td>
<td>123 4567</td>
</tr>
<tr>
<td>Joe</td>
<td>789 1234</td>
</tr>
<tr>
<td>Mary</td>
<td>584 3333</td>
</tr>
</tbody>
</table>
```
Tables

Table Styles

shade

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill</td>
<td>123 4567</td>
</tr>
<tr>
<td>Joe</td>
<td>789 1234</td>
</tr>
<tr>
<td>Mary</td>
<td>584 3333</td>
</tr>
</tbody>
</table>

Changing the Default Table Style

The default style is *columns*. This can be changed via the `DEFAULT_TABLE_STYLE` variable. For example:

```shell
!define DEFAULT_TABLE_STYLE "grid"
# the default table style is now grid ...
```

Creating New Table Styles

New table styles can be defined. See *Extending SDF* in the *SDF Guru Guide* for details.
Adding Table Titles

Adding a Title to a Table

To add a title, use the title attribute. For example:

```
!block table; title="Phone numbers of good friends"
Name  Phone
Bill   123 4567
Joe    789 1234
Mary   584 3333
!endblock
```

The output is:

Table 5-1. Phone numbers of good friends

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill</td>
<td>123 4567</td>
</tr>
<tr>
<td>Joe</td>
<td>789 1234</td>
</tr>
<tr>
<td>Mary</td>
<td>584 3333</td>
</tr>
</tbody>
</table>

Adding a Title to a Psuedo Table

Occasionally, it is useful to title a table by hand. To do this, use the TT (Table Title) paragraph style. For example:

```
TT: My Boring Looking Table
!include "my_table.txt"; example; wide
```

Generating a List of Tables

A list of tables can be generated from the table titles. To do this, set the DOC_LOT variable. For example:

```
# Build the title
!define DOC_NAME "World Peace in 60 Days"
!define DOC_LOT
!build_title
```

This will generate a list of tables after the table of contents.

Changing the List of Tables Title

The default title for the list of tables is "List of Tables". To change this, set the DOC_LOT_TITLE. For example:

```
!define DOC_LOT_TITLE "Tables"
```
Tables
Sizing Columns

Sizing Columns

**Dynamic Sizing**

SDF supports dynamic column widths for paper documentation, i.e. if a width is not specified for a column, then the column is sized based on the text within it and the space available. As a result, it is rarely necessary to explicitly size columns.

**The format Attribute**

The *format* attribute of the *table* filter is used to specify column widths for paper documentation. Its value is either:

- a single number, in which case each digit represents 10% of the width available to the table, or
- a comma-separated list of column width specifications.

For example, to specify 3 columns with widths of 20%, 10% and 30% respectively:

```
!block table; format=213
Name  Age  Address
Bill  42    123 Main Road
Joe   21    456 Queen Street
!endblock
```

The result is:

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill</td>
<td>42</td>
<td>123 Main Road</td>
</tr>
<tr>
<td>Joe</td>
<td>21</td>
<td>456 Queen Street</td>
</tr>
</tbody>
</table>

**Note:** The *format* parameter has no impact on HTML generation - all columns in HTML are currently dynamically sized.

**Column Width Specifications**

Examples of the column width specifications supported are given below.
### Tables

#### Sizing Columns

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>30pt</td>
<td>an exact size (other supported units are cm, mm, &quot; and in)</td>
</tr>
<tr>
<td>30%</td>
<td>a percentage of the size available</td>
</tr>
<tr>
<td>30</td>
<td>a percentage of the size available (% is implicit)</td>
</tr>
<tr>
<td>10-20</td>
<td>dynamic size between 10% and 20% of the total width</td>
</tr>
<tr>
<td>-20</td>
<td>dynamic size between 0% and 20% of the total width</td>
</tr>
<tr>
<td>10-</td>
<td>dynamic size between 10% and 100% of the total width</td>
</tr>
<tr>
<td>-</td>
<td>dynamic size between 0% and 100% of the total width</td>
</tr>
<tr>
<td>3*</td>
<td>3 units of the remaining space</td>
</tr>
<tr>
<td>*</td>
<td>same as 1*</td>
</tr>
</tbody>
</table>

For example, in the table below, the second column will be twice the size of the last column.

```plaintext
!block table; format="20,2*,10,*"
Name    Column2             Column3     Column4
A       B                   C           D
X       Hello dear world    Y           Z
!endblock
```

The output is:

<table>
<thead>
<tr>
<th>Name</th>
<th>Column2</th>
<th>Column3</th>
<th>Column4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>X</td>
<td>Hello dear world</td>
<td>Y</td>
<td>Z</td>
</tr>
</tbody>
</table>

### Default Sizing Rules and Narrow Tables

If a column is not given a size, the following rules are used:

1. The last unspecified column size is implicitly '*' (i.e. the rest), unless the `narrow` attribute is set, in which case the size is implicitly '-' (i.e. as much as needed).

2. The other unknown sizes are implicitly '-'.

For example, the first and third columns in the table below will be dynamically sized. The first column will take as much space as required and the last column will expand so that the table takes the full width of the text area.
Tables
Sizing Columns

!block  table; format=",30,,10"
Name  Column2       Column3       Column4
A  B               C             D
X  Hello dear world  Y             Z
!endblock

The output is:

<table>
<thead>
<tr>
<th>Name</th>
<th>Column2</th>
<th>Column3</th>
<th>Column4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>X</td>
<td>Hello dear world</td>
<td>Y</td>
<td>Z</td>
</tr>
</tbody>
</table>

However, in the example below, each column will only take as much space is required, making the table narrower than it would be otherwise.

!block  table; narrow
Name  Column2       Column3       Column4
A  B               C             D
X  Hello dear world  Y             Z
!endblock

The output is:

<table>
<thead>
<tr>
<th>Name</th>
<th>Column2</th>
<th>Column3</th>
<th>Column4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>X</td>
<td>Hello dear world</td>
<td>Y</td>
<td>Z</td>
</tr>
</tbody>
</table>

Equalised Column Widths
If an = character is used in place of a - character for a column width, then those columns will be equalised in size. For example, the second and forth columns in the table below will be made equal in size.

!block  table; format="20,5=30,10,=
Name  Column2       Column3       Column4
A  B               C             D
X  Hello dear world  Y             Z
!endblock

The output is:

<table>
<thead>
<tr>
<th>Name</th>
<th>Column2</th>
<th>Column3</th>
<th>Column4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>X</td>
<td>Hello dear world</td>
<td>Y</td>
<td>Z</td>
</tr>
</tbody>
</table>
### Tables

**Sizing Columns**

<table>
<thead>
<tr>
<th>Name</th>
<th>Column2</th>
<th>Column3</th>
<th>Column4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>X</td>
<td>Hello dear world</td>
<td>Y</td>
<td>Z</td>
</tr>
</tbody>
</table>
Positioning Tables

Horizontal Alignment

The horizontal alignment of a table can be controlled by setting the `align` parameter of the `table` filter. The permitted values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>left-align the table</td>
</tr>
<tr>
<td>Center</td>
<td>center the table</td>
</tr>
<tr>
<td>Right</td>
<td>right-align the table</td>
</tr>
<tr>
<td>Inner</td>
<td>align the table with the inner margin</td>
</tr>
<tr>
<td>Outer</td>
<td>align the table with the outer margin</td>
</tr>
</tbody>
</table>

**Note**: The `wide` parameter changes the left indent of a table to include the sidehead of a page. Therefore, the `wide` parameter will impact the horizontal positioning of any table which is not right-aligned.

Vertical Positioning

The vertical placement of a table can be controlled by setting the `placement` parameter of the `table` filter. The permitted values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Float</td>
<td>next column if necessary</td>
</tr>
<tr>
<td>Pagetop</td>
<td>top of the next page</td>
</tr>
<tr>
<td>Columntop</td>
<td>top of the next column</td>
</tr>
<tr>
<td>Lefttop</td>
<td>top of the next left-hand page</td>
</tr>
<tr>
<td>Righttop</td>
<td>top of the next right-hand page</td>
</tr>
</tbody>
</table>
Creating Heading, Footing and Group Rows

Overview

The `table` filter now supports the following attributes:

- **headings** - the number of heading rows at the top of the table
- **footings** - the number of footing rows at the end of the table
- **groups** - look for group rows in the table.

Heading Rows

If the `headings` attribute is not defined, then the column headings are generated using the column names given on the `parsing` line. For example, the column headings in the table below will be *Name* and *Age*:

```
!block table
Name    Age
Bill    42
!endblock
```

Alternatively, if the `headings` attribute is defined, then that number of data rows are used as the column headings, i.e. the parsing line is not used to build the column headings. For example, the column headings in the table below will be "Title" and *Age* (remembering that a filter attribute is implicitly given the value 1 is no value is supplied).

```
!block table; headings
A       B
"Title" Age
Bill    42
Sally   23
!endblock
```

Likewise, the column headings below will be *Preferred Title* and * Likely Age* with each heading taking 2 rows.

```
!block table; headings=2
A       B
Preferred Likely
Title    Age
Bill    42
Sally   23
!endblock
```

Footing Rows

These work exactly the same as table headings, except that no footings are generated by default.

Group Rows

If the `groups` parameter is specified, SDF assumes that rows that end in : are group rows. For example:
Tables
Creating Heading, Footing and Group Rows

The result is:

<table>
<thead>
<tr>
<th>Fruit</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tropical:</strong></td>
<td></td>
</tr>
<tr>
<td>mango</td>
<td>a fruit-bat favorite</td>
</tr>
<tr>
<td>banana</td>
<td>great for smoothies</td>
</tr>
<tr>
<td><strong>Other:</strong></td>
<td></td>
</tr>
<tr>
<td>Tomato</td>
<td>often matched with basil</td>
</tr>
</tbody>
</table>

Fruit      Comments
Tropical: mango  a fruit-bat favorite
banana great for smoothies
Other: Tomato often matched with basil
Other Table Goodies

Column alignment

The colaligns parameter of the table filter can be used to control the alignment of text within columns of a table. For example:

```
!block table; colaligns="LCCR"
Name   Column2     Column3     Column4
A      B           C            1.0
X      Hello dear world  Y       10.2
!endblock
```

The output is:

<table>
<thead>
<tr>
<th>Name</th>
<th>Column2</th>
<th>Column3</th>
<th>Column4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>1.0</td>
</tr>
<tr>
<td>X</td>
<td>Hello dear world</td>
<td>Y</td>
<td>10.2</td>
</tr>
</tbody>
</table>

The value of colaligns is usually a sequence of the letters L, C and R (which mean what one would expect). If you prefer, a comma-separated list of the values Left, Center and Right can be specified. For example:

```
!block table; colaligns="Left,Center,Center,Right"
Name   Column2     Column3     Column4
A      B           C            1.0
X      Hello dear world  Y       10.2
!endblock
```

Likewise, the colvaligns parameter can be used to control the vertical alignment of text within columns of a table. The value is either:

- a comma-separated list of the values Top, Middle, Bottom and Baseline, or
- a sequence of the letters T, M, B (Bottom) and L (baseLine).

Baseline alignment means that the base of the first line of the text in each cell is aligned (HTML only).

Note: For a variety of reasons, the default vertical alignment for cells in a HTML table is Middle, while the default vertical alignment for tables in most other formats is Top. As most tables contain rows which fit on one line, this inconsistency is not usually a problem. However, if you want consistent vertical alignment for a complex table across all formats, you need to use the colvaligns parameter to override the default behaviour.
CTables
Other Table Goodies

Column tagging
In a similar way, columns can be tagged with a phrase style by using the \texttt{coltags} attribute. For example:

\begin{verbatim}
!block table; coltags="B,EMAIL"
Purpose     Email
Information  sdf-users@mincom.com
Bug reporting sdf-users@mincom.com
!endblock
\end{verbatim}

The result is:

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information</td>
<td><a href="mailto:sdf-users@mincom.com">sdf-users@mincom.com</a></td>
</tr>
<tr>
<td>Bug reporting</td>
<td><a href="mailto:sdf-users@mincom.com">sdf-users@mincom.com</a></td>
</tr>
</tbody>
</table>

Table filtering and sorting
Tables can be filtered and sorted by using the \texttt{where} and \texttt{sort} attributes of the \texttt{table} filter. These attributes are also supported by the class filters (e.g. terms, references). In either case, filtering is done before sorting.

The \texttt{where} attribute takes an expression which is evaluated for each record. Special symbols available are:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$_</td>
<td>the current record</td>
</tr>
<tr>
<td>$o{&quot;xyz&quot;}</td>
<td>the value of column xyz</td>
</tr>
<tr>
<td>$var{&quot;abc&quot;}</td>
<td>the value of variable abc</td>
</tr>
</tbody>
</table>

For example:

\begin{verbatim}
!define MODULE_CODE "XYZ"

!include ".../mydata.tbl"; table; \
 where='$o{"Code"} =~ /$var{"MODULE_CODE"}/'
\end{verbatim}

\texttt{sort} takes a comma-separated list of column names to sort on. If no columns are specified, the data is sorted using all columns in the order in which they appear. All sorting is done alphabetically - numeric sorting is not supported.

Deleting and selecting columns
The \texttt{table} filter supports the following parameters for changing the columns actually displayed:
• **delete** - a comma-separated list of columns to delete
• **select** - the comma-separated list of columns to display.

These parameters are often used in combination with the **where** parameter. For example:

```plaintext
# Display the open bugs, sorted by priority
H1: Open Defects
!include 'mybugs.tbl'; table; \
  where='o("Status") eq "Open"'; \
  sort='Priority'; \
  delete='Status'
```

If both **delete** and **select** are specified, **select** has precedence.

---

### Macros inside tables

Macros can be used inside tables. In particular:

- rows can now be conditionally included or excluded
- rows can be marked as changed.

For example:

```plaintext
!block table
Code   Description
BTW    By the way
!block changed
RTFM   Read the (fine) manual
!endblock
!if ALL_TERMS
SOS    Save our souls
!endif
!endblock
```

---

### Landscape tables

Landscape tables are supported via the **landscape** parameter of the **table** filter. The value is the height allocated to the area in which the table is placed. If a unit is not specified, the value is assumed to be a percentage of the text column height. For convenience, a value of 1 implies a full page table. Some examples are given below.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>landscape=&quot;50pt&quot;</td>
<td>height allocated to table is 50 points</td>
</tr>
<tr>
<td>landscape=&quot;50%&quot;</td>
<td>half page table</td>
</tr>
<tr>
<td>landscape=50</td>
<td>half page table (% is the default units)</td>
</tr>
<tr>
<td>landscape=1</td>
<td>full page table (1 implies 100%)</td>
</tr>
<tr>
<td>landscape</td>
<td>full page table (syntactic shorthand for above)</td>
</tr>
</tbody>
</table>
Note: This feature is currently use at your own risk. In particular, long tables and table titles confuse it badly. Furthermore, the align and placement parameters are effectively ignored for landscape tables.

Cell attributes
Cells within a table can now be given attributes by preceding the cell value with a semicolon-separated list of name-value pairs enclosed in square brackets. For example:

```plaintext
!block table; colaligns="LCCR"
Name    Column2    Column3    Column4
X       Hello dear world  Y       [bgcolor=Red]10.2
!endblock
```

The output is:

<table>
<thead>
<tr>
<th>Name</th>
<th>Column2</th>
<th>Column3</th>
<th>Column4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C [bgcolor=Green]</td>
<td>1.0</td>
</tr>
<tr>
<td>X</td>
<td>Hello dear world</td>
<td>Y [bgcolor=Red]</td>
<td>10.2</td>
</tr>
</tbody>
</table>

The cell attributes supported are given below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>General:</td>
<td></td>
</tr>
<tr>
<td>align</td>
<td>horizontal alignment (Left, Center, Right)</td>
</tr>
<tr>
<td>valign</td>
<td>vertical alignment (Top, Middle, Bottom, Baseline)</td>
</tr>
<tr>
<td>cols</td>
<td>the number of columns this cell spans (default is 1)</td>
</tr>
<tr>
<td>rows</td>
<td>the number of rows this cell spans (default is 1)</td>
</tr>
<tr>
<td>bgcolor</td>
<td>background colour of cell (see below)</td>
</tr>
<tr>
<td>PS only:</td>
<td></td>
</tr>
<tr>
<td>fill</td>
<td>background colour fill percentage</td>
</tr>
<tr>
<td>truling</td>
<td>ruling setting for top of cell</td>
</tr>
<tr>
<td>bruling</td>
<td>ruling setting for bottom of cell</td>
</tr>
<tr>
<td>lruling</td>
<td>ruling setting for left of cell</td>
</tr>
<tr>
<td>rruling</td>
<td>ruling setting for right of cell</td>
</tr>
<tr>
<td>angle</td>
<td>angle of text (0, 90, 180, 270)</td>
</tr>
</tbody>
</table>
For PS (i.e. MIF) generation, the supported colour values are **Black, White, Red, Green, Blue, Yellow, Cyan** and **Magenta**. If a different colour is specified, it is ignored. The supported fill values are 100, 90, 70, 50, 30, 10 and 3. If a fill value is not specified, 100% fill is used.

For HTML generation, any of the HTML colours names (including those supported for PS generation) or the "#rrggbb" form can be used.

The permitted ruling values are **Vthin, Thin, Medium, Thick** and **Double**.

The *sdf*, *tag* and *paratag* attributes control the way in which the cell text is converted to SDF:

1. If *sdf* is set, the cell text is already SDF.
2. Otherwise if *tag* is set, the SDF paragraph is `paratag:{tag:text}`.
3. Otherwise, the paragraph is `paratag:text`.

*tag* is usually set via the *tags* or *groups* parameters of the *table* filter.

**Note:** *paratag* is not yet implemented.
## Spreadsheet calculations

### Overview
Spreadsheet style calculations have been introduced into SDF using the standard \([/\)]\) syntax with a prefix of + (or =) indicating that the expression is to be evaluated by the calculation routines.

This extension has been loosely modelled on Microsoft Excel® in terms of the initial functions supported and the syntax used.

The spreadsheet expression evaluator and the documentation below was written by Tim Hudson (tjh@cryptsoft.com).

### Cells and Cellids
Each cell in a table has an cellid which is made up of a single uppercase letter indicating the column index and a number indicating the row index (counting from 1 and excluding the heading rows). The upper left cell is hence A1.

An example grid indicating cellids:

<table>
<thead>
<tr>
<th>Title1</th>
<th>Title2</th>
<th>Title3</th>
<th>Title4</th>
<th>Title5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>C1</td>
<td>D1</td>
<td>E1</td>
</tr>
<tr>
<td>A2</td>
<td>B2</td>
<td>C2</td>
<td>D2</td>
<td>E2</td>
</tr>
<tr>
<td>A3</td>
<td>B3</td>
<td>C3</td>
<td>D3</td>
<td>E3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>A100</td>
<td>B100</td>
<td>C100</td>
<td>D100</td>
<td>E100</td>
</tr>
</tbody>
</table>

A range of cellids is specified using the syntax cellid1:cellid2. For example: A1:C1 is exactly the same as A1,B1,C1

### Spreadsheet Expressions
An expression consists of a combination of standard Perl operators and spreadsheet functions and cellids or cellid ranges.

Standard Perl operators include:

- + - * /

### Spreadsheet Functions
Spreadsheet functions use the syntax\n
\[\text{FUNCTION(ARG1,ARG2,...,ARGN)}\]
The following functions are supported:

- **AVERAGE** - the average - $\text{SUM(ARGS)}/\text{COUNT(ARGS)}$
- **SUM** - the sum of the args - same as $\text{ARG1+ARG2+...+ARGN}$
- **MIN** - the minimum argument value
- **MAX** - the maximum argument value
- **COUNT** - the number of arguments
- **PRODUCT** - the product of the args - same as $\text{ARG1*ARGN*...*ARGN}$
- **ROWSUM** - the **SUM** of all the cells in the row to the left of the current cell
- **ROWPROD** - the **PRODUCT** of all the cells in the row to the left of the current cell
- **COLSUM** - the **SUM** of all the cells in the column above the current cell
- **COLPROD** - the **PRODUCT** of all the cells in the column above the current cell

An Example

A simple example is shown below:

<table>
<thead>
<tr>
<th>Count</th>
<th>Price</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5</td>
<td>50.00</td>
</tr>
<tr>
<td>15</td>
<td>5.23</td>
<td>78.45</td>
</tr>
<tr>
<td>25.00</td>
<td>10.23</td>
<td>128.45</td>
</tr>
</tbody>
</table>

This generates the result below. (Ok, summing two prices is meaningless, but it illustrates the syntax.)

<table>
<thead>
<tr>
<th>Count</th>
<th>Price</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5</td>
<td>50.00</td>
</tr>
<tr>
<td>15</td>
<td>5.23</td>
<td>78.45</td>
</tr>
<tr>
<td>25.00</td>
<td>10.23</td>
<td>128.45</td>
</tr>
</tbody>
</table>

Accessing Spreadsheet Values in Paragraph Text

Values are available until the next table is processed so you can refer to data inside normal paragraphs after the table like this [[=A1]] (which evaluates to 10.00).
## Tables
Spreadsheet calculations

| Recursive Expression Evaluation | A spreadsheet expression will recursively evaluate any expressions contained in *cells* that are used in an expression. In the example above, the expression in *cell C3* depends on the results of the expression in *cell C1* and *C2*. |
| Disabling Calculations         | Calculation support for a table can be disabled by adding in an attribute of *nocalcs*. (Without this, the pointers required to table data that are needed when doing spreadsheet calculations occur for each table cell.) |
Chapter 6. SDF Advanced Features

Introduction

This chapter presents information on some advanced SDF features. SDF beginners should skip this chapter and return to it if and when they wish to learn more.

The topics covered are outlined below.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modules and Libraries</td>
<td>how to load modules and libraries</td>
</tr>
<tr>
<td>Object Management</td>
<td>how to define and use objects (e.g. references)</td>
</tr>
<tr>
<td>Conditional Text</td>
<td>how to conditionally include or exclude SDF</td>
</tr>
<tr>
<td>Event Processing</td>
<td>how to change the layout and automate tasks using rules</td>
</tr>
<tr>
<td>Custom Formatting</td>
<td>how to change fonts, paper sizes, etc.</td>
</tr>
<tr>
<td>Hypertext</td>
<td>how to build hypertext</td>
</tr>
<tr>
<td>Escaping Symbols</td>
<td>how to include special symbols (e.g. {{ })</td>
</tr>
<tr>
<td>Multiple Looks</td>
<td>packaged document-wide presentation styles</td>
</tr>
</tbody>
</table>
Modules and Libraries

Modules

SDF can be extended by using modules. A module is a normal SDF file which typically contains reusable entities like macros, filters and styles. Modules are loaded by the use macro. For example:

!use "delphi"

By convention, modules are given the sdm extension. The use macro adds this extension if none is provided. Modules can also be loaded by using sdf's -u option.

Libraries

A collection of related modules can be placed into a library which is simply a subdirectory under the sdf/home directory on Perl's library path. Libraries can be loaded by the inherit macro which essentially does the following:

1. appends the matching directory onto the include path
2. loads the matching main module.

For example:

!inherit "images"

1. appends sdf/home/images onto the include path
2. loads the sdf/home/images/images.sdm module.

The Configuration Library

After loading the standard library (stdlib), SDF loads the configuration library, if any. This is typically specified via the OPT_CONFIG variable being initialised on the top line of the file via the init macro. It can also be set via sdf's -c option.

While sdf only supports a single configuration library, additional libraries can be loaded via the inherit macro. Furthermore, a library can inherit things from other libraries, making it easy to create new ones.
Object Management

Overview

Object management is a collection of features which support:

1. the definition of special objects in configuration files
2. the use of these special objects in documents.

For example, a configuration file can contain a lookup table of references. Within a document:

- a table of references can be inserted by simply specifying document codes - the document names are found from the lookup table
- REF/DOC object phrases are validated as legal references/documents
- hypertext is generated for:
  - entries in the table of references
  - REF and DOC object phrases
- REF objects can be converted to DOC objects and visa versa.

SDF supports 4 predefined classes of objects - references, terms, products and organisations. New classes can be added via the class macro.

The Predefined Classes

The fields supported by the predefined classes are given below.

<table>
<thead>
<tr>
<th>Class</th>
<th>Styles</th>
<th>Identifier fields</th>
<th>Property fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>references</td>
<td>REF, DOC</td>
<td>Reference, Document</td>
<td>Jump, Version,</td>
</tr>
<tr>
<td>terms</td>
<td>TERM</td>
<td>Term, Definition</td>
<td>Status</td>
</tr>
<tr>
<td>products</td>
<td>PRD</td>
<td>Name</td>
<td>Jump</td>
</tr>
<tr>
<td>organisations</td>
<td>ORG, ORGN</td>
<td>Name, Long</td>
<td>Jump</td>
</tr>
</tbody>
</table>

Styles are the tags used for objects in the class. The first style is the 'normal' one. The second style is the 'long' one.

Identifier fields are the 'normal' and 'long' (if any) fields within the data tables. Property fields are the additional fields, if any.
Note: The build_title macro uses the references declarations to provide default values for DOC_CODE, DOC_VERSION, DOC_STATUS and DOC_URL.

The objects module provides some commonly used objects in these predefined classes (and matching variables).

### Declaring Objects

To declare a table of objects, the syntax is:

```plaintext
!block class; data
field names 
object record1
... 
object recordN
!endblock
```

An example lookup table for organisations is given below.

```plaintext
!block organisations; data
Name        Long                             Jump
DEC         Digital Equipment Corporation    http://www.dec.com
Frame       Frame International              http://www.frame.com
HP          Hewlett Packard                  http://www.hp.com
IBM         International Business Machines  http://www.ibm.com
Mincom      Mincom Pty Ltd                   http://www.mincom.com
Microsoft   Microsoft Corporation            http://www.microsoft.com
!endblock
```

To insert a table of objects, the syntax is:

```plaintext
!block class
field names
object key or record1
... 
object key or recordN
!endblock
```

If only the key is provided, then the 'long' name is found from the configured data, if any. For example, a configuration file may have the following declaration of terms:

```plaintext
!block terms; data
Term     Definition
SDF      Simple Document Format
SQL      Structured Query Language
!endblock
```

These can then be inserted into a document as follows:
Hypertext and Name Conversion

If an object has a 'Jump' property defined for it, hypertext is automatically generated for it. For example:

See {{DOC:My Design Document}} for further details

will have a jump created for the DOC object.

Object names can be converted between the normal and long forms by using the expand and shrink attributes. For example:

{{PRD:Java}} was developed by {{ORG[expand]Sun}}.
Conditional Text

Blocks of text can be conditionally included or excluded using the following macros:

!if condition
!elsif condition
!else
!endif

These macros allow you to tune the output for different audiences. For example, you may wish to only include email directions if HTML is being generated:

!if OPT_PP_HTML
Click {{[jump="mailto:joe"] here}} to send me email.
!endif

Note:

1. elseif can be used instead of elsif if you prefer.
2. condition is an SDF expression where unknown variables do not give errors
3. Conditional text blocks must be fully defined within a given file. This prevents an if macro within an included file excluding text in the file doing the including.
Event Processing

The on Macro

The on macro allows you to execute an arbitrary piece of Perl code during document conversion. The syntax of the on macro is:

!on type pattern; [id]; action

where:

- type specifies the event type
- pattern is a Perl regular expression string, anchored at both ends, which specifies which styles (or names) the action is to be executed for
- id is optionally used to name an event for later disabling via the off macro
- action is the block of Perl code to execute.

For example, the following statement makes every heading a hypertext target named itself:

!on paragraph 'H\d';; $attr{'id'} = $text

Event Types

The types supported and the symbols available in the respective actions include:

<table>
<thead>
<tr>
<th>Type</th>
<th>Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>paragraph</td>
<td>$style, $text, %attr</td>
</tr>
<tr>
<td>phrase</td>
<td>$style, $text, %attr</td>
</tr>
<tr>
<td>macro</td>
<td>$name, $args</td>
</tr>
<tr>
<td>filter</td>
<td>$name, $params</td>
</tr>
<tr>
<td>table</td>
<td>$style, %param</td>
</tr>
</tbody>
</table>

Event Patterns

Some example event patterns are given below.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>'XYZ'</td>
<td>matches a thing called XYZ</td>
</tr>
</tbody>
</table>
If the pattern is an empty string, the action is executed for all entities of that type.

If multiple actions are registered for a given type, actions are executed in "last in, first out" order.

The off Macro

The off macro is used to cancel an event. The syntax is:

!off type id

For example:

!on paragraph 'H\d'; XYZ; $attr{'id'} = $text
# lots of SDF
!off paragraph XYZ

If two events are given the same name, the most-recently-named event will be cancelled. i.e. nested event cancelling works as expected.

Examples

Generating Hypertext Targets

To make all level 1 and 2 headings hypertext targets:

!on paragraph '[(HAP)\d]';; $attr{'id'} = $text

Generating Index Entries

To make index entries for all commands (CMD character tags):

!on phrase 'CMD';; $attr{'index'} = $text

Adjusting Heading Levels

To move normal headings down one level:

!on paragraph 'H\d';; $style =~ tr/1234/2345/

Changing Spelling

To convert selected words to North American spelling:

!on paragraph '';; $text =~ s/colour/color/ig; \ $text =~ s/ise\b/ize/g

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>'AB</td>
<td>CD</td>
</tr>
<tr>
<td>'H1'</td>
<td>for paragraphs, matches a level 1 normal heading</td>
</tr>
<tr>
<td>'H[1-4]'</td>
<td>for paragraphs, matches normal headings at levels 1 to 4</td>
</tr>
<tr>
<td>'[HAP]\d'</td>
<td>for paragraphs, matches all headings</td>
</tr>
</tbody>
</table>
This approach uses Perl's substitute operator on the text in each paragraph:

- the 'i' switch means case insensitive
- the 'g' switch means global, i.e. all occurrences in each paragraph
- the '\b' symbol matches a word boundary.
Custom Formatting

Overview

One of the nice things about SDF is that system administrators can control the formatting of documents via templates. This has two major advantages:

1. document authors can focus on content
2. by default, documents have a consistent format.

However, there are occasions when authors need to override the default formatting conventions. Custom formatting is supported for documents, paragraphs and phrases.

Custom Document Formatting

Document formatting can be controlled by setting variables. The commonly used ones are given below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOC_TOC</td>
<td>integer</td>
<td>table of contents level</td>
</tr>
<tr>
<td>DOC_LOF</td>
<td>boolean</td>
<td>build a list of figures</td>
</tr>
<tr>
<td>DOC_LOT</td>
<td>boolean</td>
<td>build a list of tables</td>
</tr>
<tr>
<td>DOC_IX</td>
<td>boolean</td>
<td>build an index</td>
</tr>
<tr>
<td>DOC_TWO_SIDES</td>
<td>boolean</td>
<td>build a two-sided document</td>
</tr>
</tbody>
</table>

For example, to create a two-sided document, add this line to your document:

```
!define DOC_TWO_SIDES
```

Custom Paragraph Formatting

Paragraph formatting can be controlled by setting attributes. The commonly used ones are given below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>prefix label (e.g. Note:)</td>
</tr>
<tr>
<td>top</td>
<td>either Page (the default), Column, LPage or RPage</td>
</tr>
<tr>
<td>left</td>
<td>left margin indent (in points)</td>
</tr>
<tr>
<td>first</td>
<td>first line indent (in points)</td>
</tr>
<tr>
<td>right</td>
<td>right margin indent (in points)</td>
</tr>
<tr>
<td>justify</td>
<td>either Full, Left, Right or Center</td>
</tr>
</tbody>
</table>
For example, to start a paragraph at the top of a page:

```
H3[top] Other Issues
```

To indent a paragraph 1 inch from both margins:

```
[left='1in'; right='1in'] This paragraph will be indented from both margins by 1 inch.
```

The result is:

This paragraph will be indented from both margins by 1 inch.

---

**Custom Phrase Formatting**

Phrase formatting can be controlled by setting attributes. The commonly used ones are given below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>family</td>
<td>font family</td>
</tr>
<tr>
<td>size</td>
<td>font size</td>
</tr>
<tr>
<td>color</td>
<td>text colour</td>
</tr>
<tr>
<td>bold</td>
<td>enable bold</td>
</tr>
<tr>
<td>italics</td>
<td>enable italics</td>
</tr>
<tr>
<td>underline</td>
<td>enable underline</td>
</tr>
<tr>
<td>changed</td>
<td>enable change bars</td>
</tr>
</tbody>
</table>

For example, to change the size of a phrase to 11.5 points:

```
I love {{N[size=11.5]custom formatting}}.
```

The result is:

I love custom formatting.

---

**Applying Phrase Attributes to Paragraphs**

Phrase attributes can also be used for paragraphs. For example, to add a change bar to a paragraph:

```
[changed] This paragraph has a change bar.
```

The result is:

This paragraph has a change bar.
sdf Blocks

Attributes can be applied to a block of text by using the *sdf* filter. For example:

```
!block sdf; size='8pt'; bold; align='Center'
This paragraph has custom formatting:

* and so does this.
!endblock
```

The result is:

```
This paragraph has custom formatting:

• and so does this.
```
Hypertext

Attributes

Hypertext can be specified using the id and jump phrase attributes. The value of id is a text string which can include letters, digits, underscores and spaces. The value of jump is a World Wide Web (WWW) Uniform Resource Locator (URL).

Commonly used jump formats are summarised below.

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#xyz</td>
<td>jump to xyz within this document</td>
</tr>
<tr>
<td>abc</td>
<td>jump to document abc</td>
</tr>
<tr>
<td>abc#xyz</td>
<td>jump to xyz within document abc</td>
</tr>
</tbody>
</table>

For example:

This is the only software you will ever need.
See \{CMD[jump="#smartcmd"] smartcmd\} for details.

H2\{id="smartcmd"\} smartcmd - Do What I Want Utility

{{CMD:smartcmd}} is the best program ever written.
It detects what you want to do next by ...

If you want to specify a phrase as a hypertext target within "normal" paragraph text, use the N (for Normal) phrase style. For example:

The {{N[id='_option_o']}-o option} specifies ...

Note: Hypertext jumps are ignored when paper documents are being generated.
Escaping Symbols

Special Characters at the Start of a Line

To escape a special character at the start of a line, precede it with a backslash character (\).

Tag Delimiters

Like special characters, any special pattern at the start of a line can be escaped by preceding it with a backslash character (\).

Consider the following example:

```
Important:
```

This paragraph has a tag called *Important* and has no text. *sdf* will warn you if it finds an unknown tag so most unintentional errors like this are detected. You can escape the pattern like this:

```
\Important:
```

Other workarounds are:

```
:Important:
N:Important:
```

Special Symbols

SDF uses a number of special symbols inside paragraphs:

- A-Z < and > - delimit *concise* phrases
- {{ and }} - delimit *verbose* phrases
- [[ and ]] - delimit expressions (e.g. variables).

If you need to include one of these symbols in a paragraph, use the appropriate escape as shown below.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Escape</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>E&lt;lt&gt;</td>
</tr>
<tr>
<td>&gt;</td>
<td>E&lt;gt&gt;</td>
</tr>
<tr>
<td>{{</td>
<td>E&lt;2{&gt;</td>
</tr>
<tr>
<td>}}</td>
<td>E&lt;2&gt;}</td>
</tr>
<tr>
<td>[[</td>
<td>E&lt;2[&gt;</td>
</tr>
<tr>
<td>]]</td>
<td>E&lt;2&gt;</td>
</tr>
</tbody>
</table>

Note the following rules:

1. It isn't necessary to escape any symbols within:
   - paragraphs tagged with the V style or > tag
   - paragraphs given the verbatim attribute
   - paragraphs within a verbatim block
2. It is only necessary to escape a `<` when it appears after a capital letter.
3. It is only necessary to escape a `>` inside a concise phrase. In particular, `>` is implicitly escaped within a verbose phrase.

**Phrase Parsing Rules**

For paragraphs which do not have the *verbatim* attribute set, the parsing rules for the paragraph text are:

1. Expressions embedded in `[[` and `]]` are recursively expanded.
2. Verbose phrases embedded in `{{` and `}}` are recursively expanded to concise phrases with a V tag.
3. Concise phrases are parsed.

**Multi-line Macros**

!-style macros can be continued onto the next line by ending the line with a backslash character (`\`). To escape this backslash, use another one. If N backslash characters are found at the end of a line where N is greater than 2, then that line is terminated by N-1 backslash characters and the macro is continued onto the next line.

**Embedded Expressions**

As expressions embedded within text are terminated by the `]]` symbol, expressions cannot contain this character sequence. If you need this sequence within an expression, there is usually a simple workaround you can use. For example:

1. use a space character (i.e. `$a[$b[1]]` becomes `$a[$b[1] ]`)
2. use string concatenation (i.e. `"]]"` becomes `"]"."]"`).

**Semi-Colons Within Attributes**

If you need to include a real semi-colon within an attribute or parameter, use two semi-colons. In general, if you need to include a sequence of N semi-colons in an attribute expression, use N+1 semi-colons. For example:

To install `{{P[index="myApp;;install:myApp"]myApp}}`, ...

In this case, the value of *index* is `myApp:install:myApp`. 
Looks

Changing the look

The overall appearance of a paper-based output format can be controlled by initialising the OPT_LOOK variable on the top line of a document. For example:

```
!init OPT_LOOK="infomap"
```

Alternatively, sdf's -k option can be used. For example:

```
sdf -2ps -kinfomap mydoc
```

The -k option overrides the init macro setting.

**Note:** At this time (January 98), multiple looks are currently only supported for paper documentation generated via FrameMaker. Multiple look support will hopefully be added to other SDF output drivers during 1998.

Available looks

The available looks include:

- **simple** - the default look, useful for general documentation
- **fancy** - a look suitable for user documentation
- **infomap** - a look based on Information Mapping™
- **overhead** - a look suitable for overhead transparencies.

Creating new looks

It is relatively simple to create new looks by inheriting details from an existing one.

Refer to the *SDF Guru Guide* for details.
Appendix A. Credits

SDF has taken several years to put together and would not exist in its current form without the contributions below.

Firstly, thanks to my former Leeds and Northrup Australia colleagues, particularly Tom Beale, Craig Gibbings and Greg Birnie, for inspiring my initial interest in literate programming.

Secondly, thanks to Tim Hudson (tjh@cryptsoft.com) for being SDF’s biggest fan. Tim has assisted in design, fixed bugs, added features, "sold" SDF to many of its current user base and generally hassled me to package SDF into its current form. Thanks also to Chris Moran (chris.moran@mincom.com) for maintaining the SDF Home Page, assisting in design, etc. Many other Mincom colleagues have assisted with SDF, particularly Craig Willis and David Cox. Thanks to everyone involved.

SDF is built on Perl, a freely available scripting language created by Larry Wall (larry@wall.org). Without Perl, SDF would not exist. Thanks Larry.

Finally, thanks to my wife, Geraldine, for being my best friend and inspiring me to develop SDF.
Index

Symbols
4-1
"hello $planet!
" 3-4
"Title" 5-11
"User Guide" 2-14
# 6-13
# Output a nice message 3-4
+ 5-18
.log 1-4, 1-5, 1-6
.out 1-4, 1-5, 1-6
= 5-18
5-18

Numerics
213 2-14

A
A1 5-18
  C1 5-18
A1,B1,C1 5-18
abc 6-13
Acrobat ii
action 6-7
Address 5-6
Administration
  1-13
administration 1-13, 1-15
Adobe ii
Adobe Distiller 1-12
Age 5-6, 5-11
Align 4-6
align 4-6, 4-7, 5-10, 5-16
Appendices 3-1
appendix 2-11, 2-12, 3-1
Attributes 3-2
attributes 2-13
autoexec.bat 1-5, 2-5
AVERAGE 5-19

B
Baseline 5-13
before 2-9
bin/dos 1-6
bindos 1-3, 1-4, 1-5
bitmap 4-3
Black 5-17
block 1-10, 2-11
Blue 5-17
Bottom 5-13
Bug reporting 5-14
build_title 1-9, 1-13, 1-15, 1-16, 3-2, 4-2, 6-4

C
C 5-13
c
  sdf 1-3, 1-4, 1-5
  sdfbin 1-4
  sdfbindos 1-3
  sdfperllib 1-4
C1 5-20
C2 5-20
C3 5-20
cell 5-18, 5-20
cellid 5-18
cellid1
  cellid2 5-18
cells 5-20
Center 5-13
chris.moran@mincom.com A-1
Class 6-3
class 2-6, 6-3, 6-4
Classes
  2-6
classes 6-3
clear 4-7
colaligns 5-13
colaligns 5-13
COLPROD 5-19
COLSUM 5-19
coltags 5-14
Index-2

Column2 5-7, 5-8, 5-9, 5-13, 5-16
Column3 5-7, 5-8, 5-9, 5-13, 5-16
Column4 5-7, 5-8, 5-9, 5-13, 5-16
columns 5-4
colvaligns 5-13
comment lines 2-2
Comments 5-12, 6-7
Concept 2-1
concise 6-14
condition 6-6
Conditional Text 6-1
Control Panel 1-3
COUNT 5-19
Count 5-19
Custom Formatting 6-1
custom formatting 6-11
Cyan 5-17

D
data 5-11
default 2-8, 2-9
DEFAULT_TABLE_STYLE 5-4
define 1-9, 2-8, 2-9
delete 5-15
Delphi 3-4
Description 1-16, 2-1, 2-3, 2-5, 2-7, 2-8, 2-11, 4-2, 4-6, 6-1, 6-13
doc/paper 1-8
DOC_AUTHOR 2-8
DOC_LOF 4-5
DOC_LOT 5-5
DOC_NAME 2-8
DOC_TOC 2-8, 3-1
DOC_TYPE 2-8
Document styles 1-15
Double 5-17

E
elseif 6-6
elsif 6-6
Email 5-14
Emphasis
  2-5
emphasis 1 2-5
emphasis 2 2-5
emphasis 3 2-5
endblock 1-10, 2-11
endmacro 2-7
Environment 1-3
Escaping Symbols 6-1
Event Processing 6-1
example 2-11, 3-3, 3-4
Excel 5-18
execute 2-11
expand 6-5
expression 2-7, 2-9
Extending SDF 5-4

F
factotum 1-2
factotum@mincom.com 1-2
fancy 6-16
FAQ 1-2
field names 6-4
file 2-7
Filter 2-11
filter 2-7, 2-11
filters 6-2
fmbatch 1-11
footings 5-11
Format 6-13
format 5-6
Formatting 2-5
FrameMaker ii
FrameViewer ii
Fruit 5-12
ftp
  //ftp.mincom.com/pub/mtr/sdf 1-2
FUNCTION(ARG1,ARG2,...ARGN) 5-18

G
General
  1-13, 5-16
general 1-13, 1-15
Green 5-17
Greeting 3-3
groups 5-11, 5-17
gunzip 1-3

H
headings 5-11
Hello world 2-5
HTML 1-1, 6-6
HTML only
5-17
http
/www.mincom.com 2-5
/www.mincom.com/mtr/sdf 1-2
/www.mincom.com/mtr/sdf/download.html 1-3
/www.perl.com/index/ 1-3
Hypertext 6-1

I
ianc@mincom.com 2-5
id 6-7, 6-8, 6-13
Identifier fields 6-3
Identifier fields 6-3
if 6-6
import 4-1, 4-5, 4-6
Important 6-14
include 2-11
include path 6-2
index 6-15
infomap 6-16
Information 5-14
inherit 4-2, 6-2
init 1-13, 2-8, 2-9, 2-10, 6-2, 6-16
install
  myApp 6-15
Installation Notes 1-3

J
Java 3-4
jump 6-13

L
L 5-13
label 3-9
landscape 5-15
lang 3-4
larry@wall.org A-1
LaTeX 1-1
Leeds and Northrup Australia A-1
Left 5-13
length("User Guide") 2-14
library 6-2
Likely Age 5-11
listitem 4-6
literate programming A-1
local 3-4
logical markup language 3-2

M
Macro 2-7
macro 2-7
macros 2-2, 6-2
Magenta 5-17
main module 6-2
MAX 5-19
Meaning 1-9, 5-7, 5-10, 5-14, 5-15
Medium 5-17
Microsoft ii, 5-18
Middle 5-13
MIMS 2-6
MIN 5-19
Mincom 2-6, A-1
Miscellaneous
  1-13
miscellaneous 1-13, 1-15
modules 6-2
Modules and Libraries 6-1
MTR-SDF-0002 2-6
Multiple Looks 6-1
must ii
myApp 6-15
mydoc.hlp 1-12
mydoc.hpj 1-12
mydoc.html 1-11
mydoc.mif 1-11
mydoc.out 1-11
mydoc.out.ps 1-11
mydoc.ps 1-11
mydoc.rtf 1-12
mydoc.sdf 1-8, 1-11, 1-12
mydoc.txt 1-12
mydoc_1.html 1-11
mydoc_2.html 1-11
myfile.sdf 2-5

N
Name 2-13, 5-3, 5-4, 5-5, 5-6, 5-7, 5-8, 5-9,
      5-11, 5-13, 5-16, 6-10, 6-11
narrow 5-7
nocalcs 5-20
normal 5-19
not 4-1
Note 3-9
note 3-9, 3-10
notoc 3-2

O
object key or record1 6-4
object key or recordN 6-4
Object Management 2-6, 6-1
Object management 6-3
object record1 6-4
object recordN 6-4
objects 6-4
off 6-7, 6-8
on 6-7
only 2-9
OPT_CONFIG 6-2
OPT_LOOK 2-8
OPT_STYLE 2-8
Option 2-11
Options
   2-8
ordered 3-6
organisations 6-3
Other
   5-12
overhead 6-16

P
Paragraphs 2-5
      2-13
4-1
parameters 2-7
paratag 5-17
parsing 5-11
Pattern 6-7
pattern 6-7
PDF ii
Perl 1-1, 2-14, 3-4, 6-7, A-1
perl 1-7
perl -v 1-7
Phone 5-3, 5-4, 5-5
phrase 2-5
Phrases
      2-13
pkunzip 1-5
placement 5-10, 5-16
plain 2-11, 2-12, 3-1, 3-6
pod2ps 1-1
Position 4-6
position 4-6
PostScript ii, 1-1
Preferred Title 5-11
Price 5-19
print 3-4
PRODUCT 5-19
products 6-3
Property fields 6-3
PS only
      5-16
Purpose 1-13, 2-13, 5-14, 6-10, 6-11

R
R 5-13
Red 5-17
references 6-3, 6-4
Right 5-13
ROWPROD 5-19
ROWSUM 5-19
S
Sample Output 2-5
SDF ii, v, 1-2, 2-5, 2-7, 2-9, 2-11, 2-14, 3-2, 3-6, 6-1, 6-2, 6-3, 6-10, 6-14
sdf 1-4, 1-5, 1-6, 1-7, 1-11, 1-13, 2-8, 2-9, 3-4, 5-17, 6-2, 6-12, 6-14, 6-16
SDF expression 6-6
SDF Guru Guide v, 1-14, 2-7, 5-4, 6-16
SDF Home Page A-1
SDF Quick Reference v
SDF Reference v, 1-15, 1-16, 2-7, 2-11, 5-2
SDF Sample Showcase 1-7
SDF User Guide 2-6
sdf.ini 1-6
sdf/home 1-6, 6-2
sdf-bugs@mincom.com v, 1-2
sdf-bugs-request@mincom.com 1-2
SDF-FAQ 4-4
sdftest 1-4, 1-5, 1-6, 1-7
sdf-users@mincom.com 1-2, 5-14
sdf-users-request@mincom.com 1-2
sdm 6-2
Section 6-1
select 5-15
SGML-Tools 1-1
shrink 6-5
simple 6-16
Sizing Columns 5-2
some bold text 2-5
some example text 2-5
some italic text 2-5
some normal text 2-5
some underline text 2-5
sort 5-14
Special
   5-17
special_tag 3-6
stdlib 6-2
stdlib.sdm 2-9
strong emphasis 2-5
Style 1-13, 2-3
style 2-2, 5-3
style category 1-13

SDF Guru Guide v, 1-14, 2-7, 5-4, 6-16
SDF Home Page A-1
SDF Quick Reference v
SDF Reference v, 1-15, 1-16, 2-7, 2-11, 5-2
SDF Sample Showcase 1-7
SDF User Guide 2-6
sdf.ini 1-6
sdf/home 1-6, 6-2
sdf-bugs@mincom.com v, 1-2
sdf-bugs-request@mincom.com 1-2
SDF-FAQ 4-4
sdftest 1-4, 1-5, 1-6, 1-7
sdf-users@mincom.com 1-2, 5-14
sdf-users-request@mincom.com 1-2
sdm 6-2
Section 6-1
select 5-15
SGML-Tools 1-1
shrink 6-5
simple 6-16
Sizing Columns 5-2
some bold text 2-5
some example text 2-5
some italic text 2-5
some normal text 2-5
some underline text 2-5
sort 5-14
Special
   5-17
special_tag 3-6
stdlib 6-2
stdlib.sdm 2-9
strong emphasis 2-5
Style 1-13, 2-3
style 2-2, 5-3
style category 1-13

SDF Guru Guide v, 1-14, 2-7, 5-4, 6-16
SDF Home Page A-1
SDF Quick Reference v
SDF Reference v, 1-15, 1-16, 2-7, 2-11, 5-2
SDF Sample Showcase 1-7
SDF User Guide 2-6
sdf.ini 1-6
sdf/home 1-6, 6-2
sdf-bugs@mincom.com v, 1-2
sdf-bugs-request@mincom.com 1-2
SDF-FAQ 4-4
sdftest 1-4, 1-5, 1-6, 1-7
sdf-users@mincom.com 1-2, 5-14
sdf-users-request@mincom.com 1-2
sdm 6-2
Section 6-1
select 5-15
SGML-Tools 1-1
shrink 6-5
simple 6-16
Sizing Columns 5-2
some bold text 2-5
some example text 2-5
some italic text 2-5
some normal text 2-5
some underline text 2-5
sort 5-14
Special
   5-17
special_tag 3-6
stdlib 6-2
stdlib.sdm 2-9
strong emphasis 2-5
Style 1-13, 2-3
style 2-2, 5-3
style category 1-13

SDF Guru Guide v, 1-14, 2-7, 5-4, 6-16
SDF Home Page A-1
SDF Quick Reference v
SDF Reference v, 1-15, 1-16, 2-7, 2-11, 5-2
SDF Sample Showcase 1-7
SDF User Guide 2-6
sdf.ini 1-6
sdf/home 1-6, 6-2
sdf-bugs@mincom.com v, 1-2
sdf-bugs-request@mincom.com 1-2
SDF-FAQ 4-4
sdftest 1-4, 1-5, 1-6, 1-7
sdf-users@mincom.com 1-2, 5-14
sdf-users-request@mincom.com 1-2
sdm 6-2
Section 6-1
select 5-15
SGML-Tools 1-1
shrink 6-5
simple 6-16
Sizing Columns 5-2
some bold text 2-5
some example text 2-5
some italic text 2-5
some normal text 2-5
some underline text 2-5
sort 5-14
Special
   5-17
special_tag 3-6
stdlib 6-2
stdlib.sdm 2-9
strong emphasis 2-5
Style 1-13, 2-3
style 2-2, 5-3
style category 1-13

T
Table 1-7
t 1-4, 1-5, 1-7
tables 1-10, 2-11, 5-1, 5-6, 5-10, 5-11, 5-13, 5-14, 5-15, 5-17
Tags 2-5
tag 1-9, 2-5
tags 5-17
tar 1-3
tar.gz 1-6
TBL 2-11, 5-1, 5-2
terms 6-3, 6-4
The SDF Document Development System v
The search order is 4-2
Thick 5-17
Thin 5-17
title 1-13, 1-15, 2-11, 4-2, 4-5, 5-5
Title1 5-18
Title2 5-18
Title3 5-18
Title4 5-18
Title5 5-18
Titles
   2-8
tjh@cryptsoft.com 5-18, A-1
Topics 2-11
top 3-2
Total 5-19
Titles
   5-12

SDF 2.001
MTR-SDF-0002.001
25-May-99
© Ian Clatworthy
Type 6-7, 6-10
  type 2-6, 6-7, 6-8
Types
  2-5
Typical result 2-5

U
  undef 2-8
  Unix ii
  unordered 3-6
  use 4-2, 6-2
  use at your own risk 5-16

V
  Value 5-7, 5-10, 5-15, 5-16
  var 2-14
  Variable 1-16, 2-8, 4-2
  variable 2-7, 2-8
  variables 2-7
  vector 4-3
  verbatim 3-3, 6-15
  verbose 6-14
  vgrind 3-4
  Vthin 5-17

W
  When generating 4-2
  where 5-14, 5-15
  White 5-17
  wide 3-3, 4-6, 5-10
  Windows ii, 1-1
  wrap_text 4-7

X
  X/Open ii
  xyz 6-13

Y
  Yellow 5-17

Z
  zip 1-6