

---

# Formatting with Box and Pandora

Paul Klint  
Taeke Kooiker  
Jurgen Vinju

2007-10-20 22:13:15 +0200 (Sat, 20 Oct 2007)

## Table of Contents

An introduction to Formatting .....	1
Why Formatting? .....	1
What is the Global Picture? .....	2
What are Box and Pandora? .....	2
How to use Box and Pandora? .....	2
Learning more .....	2
The Box Language .....	3
Box Expressions .....	3
Box Operators and Spacing Options .....	4
Fonts .....	7
Cross references .....	7
Pandora .....	7
Examples .....	8
Historical Notes .....	8
To Do .....	9
Bibliography .....	9

## Warning

This document is work in progress. See ToDo section.

## An introduction to Formatting

If you want to have a uniform style to present your

- source code of programs in existing language like C, C++, Java, or Cobol, or
- source code of programs in domain-specific languages, or
- data files in formats like XML, comma-separated lists, BibTex, or others

then you need a *formatter* that takes the original source text and applies uniform formatting rules to it. In that case **Box** and **Pandora** may be the right technologies for you.

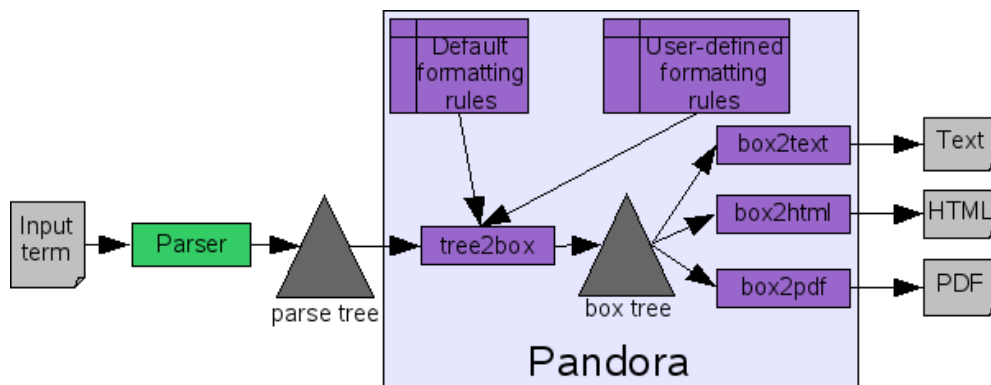
## Why Formatting?

It is well-known that readability of source code has a major impact on the productivity of software development and on the quality of the resulting software. This is particularly true when software is being developed in teams; it is then mandatory to adhere to a common coding and presentation style of the source code. Automatic formatting is a technology for achieving this uniformity.

## What is the Global Picture?

Our global formatting architecture is shown in Figure 1.1, “Formatting Architecture” (page 2). A source code program (“Input term”) is parsed and converted into a parse tree. Next, the parse tree is converted into a box expression (“box tree”) and that box expression can finally be converted into various output formats. The crucial step is from parse tree to box expression where both default formatting rules and user-defined formatting rules play a role.

**Figure 1.1. Formatting Architecture**



This architecture also explains the two alternative names for formatting: *pretty printing* (produce a new version of the source text that is prettier than the original one) and *unparsing* (parsing converts from text to parse tree, unparsing converts from parse tree to text).

## What are Box and Pandora?

Box is a language-independent intermediate language for describing various formatting aspects of source text like horizontal or vertical positioning of text fragments and their indentation, font, and color. Box can easily be converted to various textual formats like ASCII, HTML, PDF, LaTeX and others.

Pandora is a complete formatting system that takes care of converting source text to its representation in Box and producing final output. Given the grammar of a programming language it can fully automatically produce a formatter for that language using default rules. This default formatter can be adapted to specific formatting requirements by giving user-defined formatting rules that describe those requirements and overrule the default behaviour.

## How to use Box and Pandora?

There are three ways to use Box and Pandora:

- By far the simplest way is to use the ASF+SDF Meta-Environment which provides a default formatter for every grammar. This default formatter may, however, not satisfy your requirements.
- Another, equally simple way, is to write dedicated formatting rules in your language definition. For each language  $L$ , its grammar is placed in  $L/syntax$  while the formatting rules are placed in  $L/format$ . Those rules are activated automatically whenever an  $L$  source text is formatted.
- At the command line the command **pandora** can be used to convert a parse tree to a box term.

## Learning more

We will now describe:

- The Box Language.
- Pandora.
- Examples.

In the section called “Historical Notes” (page 8), we give background and key references.

# The Box Language

## Box Expressions

The general format of a Box expression is either a literal string:

```
"some text"
```

or a composite expression of the form:

```
BoxOperator SpaceOptions [ Box1 Box2 ... ]
```

Here `BoxOperator` is one of the operators listed in Table 1.1, “Box Operators” (page 3) This operator controls the formatting of the boxes `Box1`, `Box2`, ... The `SpaceOptions` are one of the options listed in Table 1.2, “Spacing and Alignment Options” (page 4) and control the amount of horizontal, vertical, or indentation space between the boxes. Some operators may have additional options. Box operators never lose operands or change the order of appearance of their operands on a page read left-to-right and top-to-bottom.

**Table 1.1. Box Operators**

Operator	Options	Description
H	hs	Horizontal formatting of sub-boxes
V	vs	Vertical formatting of sub-boxes
HV	hs, vs	Horizontal and vertical formatting of sub-boxes
HOV	hs, vs	Horizontal or vertical formatting of sub-boxes
I	is	Indented box
WD	-	Horizontal width
COMM	-	Comment
A	l, c, r, hs, vs, is	Alignment of rows in a table
R	l, c, r, hs, vs, is	Row in a table
G	gs, op	Grouping of arbitrary list elements
SL	-	Grouping of separated list

**Table 1.2. Spacing and Alignment Options**

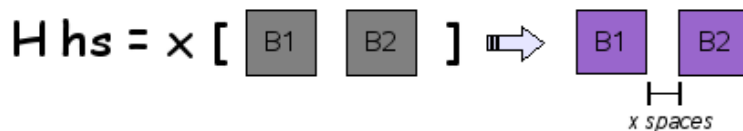
Operator	Description
hs	Horizontal spacing
vs	Vertical spacing
is	Indentation spacing
ts	Tab stop spacing
gs	Group size
l	Left-aligned
c	Center-aligned
r	Right-aligned

## Box Operators and Spacing Options

### H Box

The H box operator places sub-boxes horizontally as shown in Figure 1.2, “Horizontal Box Operator (H)” (page 4) Optionally, an horizontal spacing option (hs) can be given to indicate the desired separation between the sub-boxes.

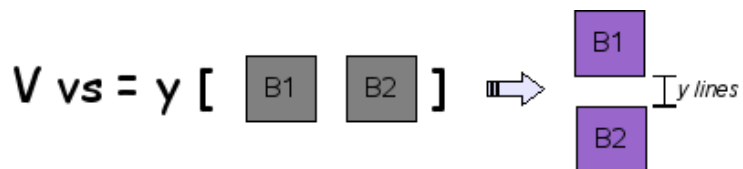
**Figure 1.2. Horizontal Box Operator (H)**



### V Box

The V box operator places sub-boxes vertically as shown in Figure 1.3, “Vertical Box Operator (V)” (page 4) Optionally, a vertical spacing option (vs) can be given to indicate the desired vertical separation between the sub-boxes.

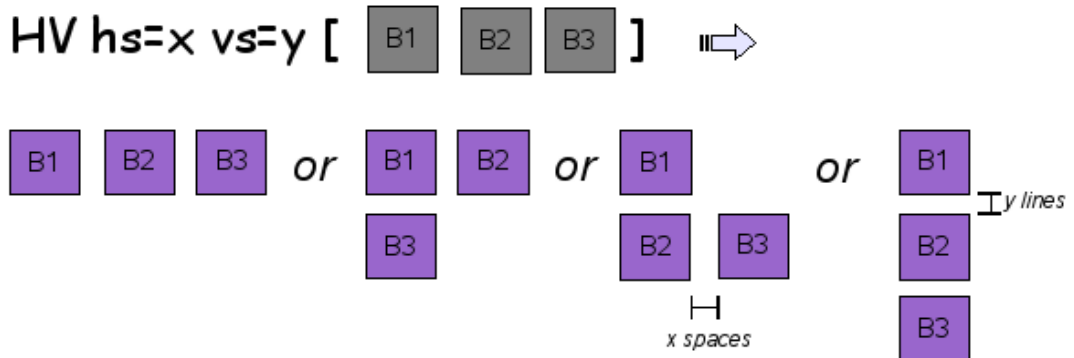
**Figure 1.3. Vertical Box Operator (V)**



### HV Box

The HV box operator places as much of its sub-boxes horizontally as possible as shown in Figure 1.4, “Horizontal/Vertical Box Operator (HV)” (page 5) Optionally, a horizontal (hs) or vertical (vs) spacing option can be given.

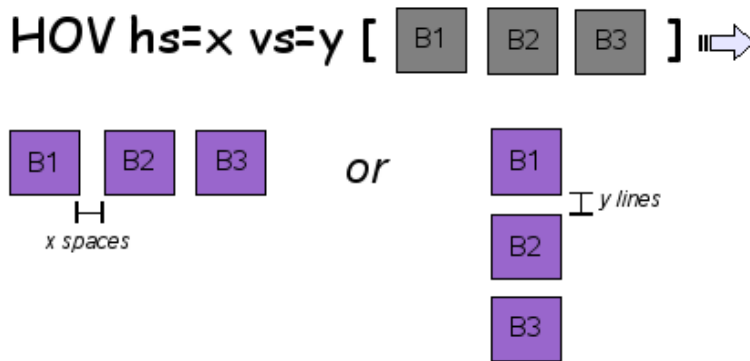
**Figure 1.4. Horizontal/Vertical Box Operator (HV)**



## HOV Box

The HOV box operator places its sub-boxes either horizontally or vertically, depending on available horizontal space. This is shown in Figure 1.5, “Horizontal or Vertical Box Operator (HOV)” (page 5). Optionally, a horizontal (*hs*) or vertical (*vs*) spacing option can be given.

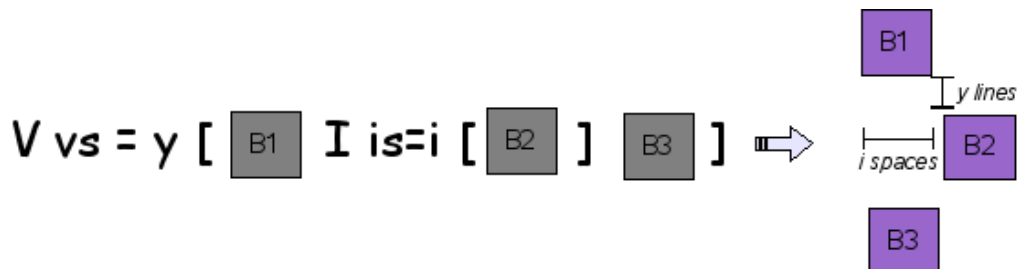
**Figure 1.5. Horizontal or Vertical Box Operator (HOV)**



## I Box

The I box operator indents its single sub-box horizontally. The effect is best illustrated in combination with a vertical box operator as shown in Figure 1.6, “Indentation Box Operator (I)” (page 5). Optionally, an indentation (*is*) spacing option can be given.

**Figure 1.6. Indentation Box Operator (I)**

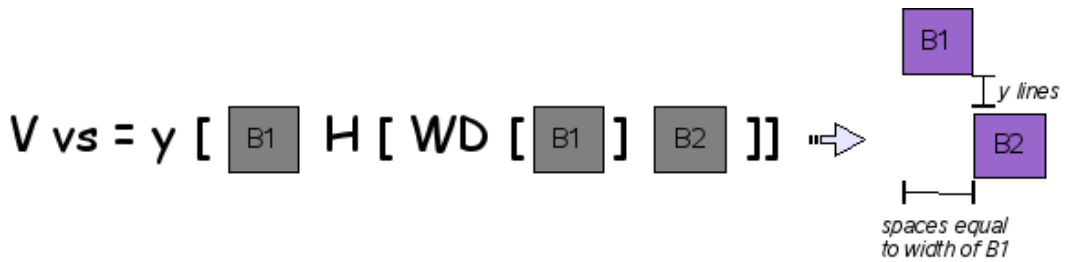


## WD Box

The I box operator defines a static amount of indentation. In some cases it is desirable to determine the indentation dynamically based on the horizontal dimensions of a given, already formatted, box.

This can be achieved with the WD box operator that creates a box with the width of its single sub-box. The effect is shown in an example in Figure 1.7, “Width Box Operator (WD)” (page 6).

**Figure 1.7. Width Box Operator (WD)**



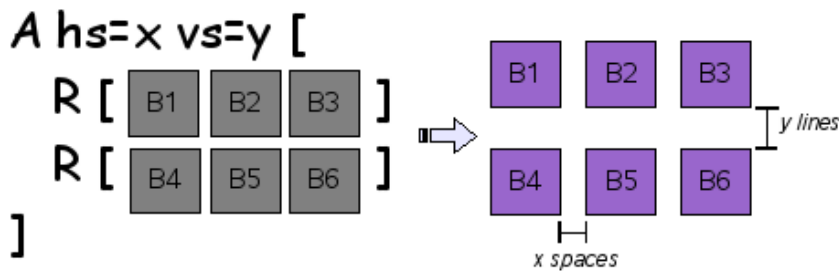
## COMM Box

The box operator COMM marks its sub-boxes as comment. It has no other effect on formatting.

## A and R Box

An A box operator declares an alignment environment in which R boxes are aligned in columns as shown in Figure 1.8, “Alignment (A) and Row (R) Box Operators” (page 6).

**Figure 1.8. Alignment (A) and Row (R) Box Operators**



## G Box

The box operator G operator is a generalization of the A and R operators: it wraps another operator around every Nth element of a list of boxes:

```
G gs=N op=OP [ Box1 Box2 ... ]
```

While a table has a fixed number of rows and columns, the G operators can be used to generate, for instance, R operators dynamically. This is useful while formatting list of arbitrary length. In such a case, the G operator takes care of chopping the list in gs elements and processing those elements. Typically, but not exclusively, they are placed in a row.

## SL Box

The SL box operator is an abbreviation for

```
G gs=4 op=OP [ Box1 Box2 ... ]
```

and is typically used for the formatting of separated lists. The four elements corresponding to gs=4 are:

- the white space before the list element;

- the list element itself;
- the white space following the list element;
- the list separator following the list element.

## Fonts

The font operators determine the font to be used for the text in their sub-boxes. The general font operator `F` allows fully general font selection. The operators `KW`, `VAR`, `NUM`, `MATH`, `ESC`, `COMM`, and `STRING` define fonts for some common cases as found in programming languages. These operators are summarized in Table 1.3, “Font Operators” (page 7) Font parameters are listed in Table 1.4, “Parameters of Font Operator `F`” (page 7).

**Table 1.3. Font Operators**

Font Operator	Font Parameters	Description
<code>F</code>	<code>fn, fm, se, sh, sz, cl</code>	General Font operator
<code>KW</code>	-	Keyword Font
<code>VAR</code>	-	Variables Font
<code>NUM</code>	-	Numbers Font
<code>MATH</code>	-	Mathematics Font
<code>ESC</code>	-	Escape Font
<code>COMM</code>	-	Comment Font
<code>STRING</code>	-	String Font

**Table 1.4. Parameters of Font Operator `F`**

Font Parameter	Description
<code>fn</code>	Font name
<code>fm</code>	Font family
<code>se</code>	Font series
<code>sh</code>	Font shape
<code>sz</code>	Font size
<code>cl</code>	Font color

## Cross references

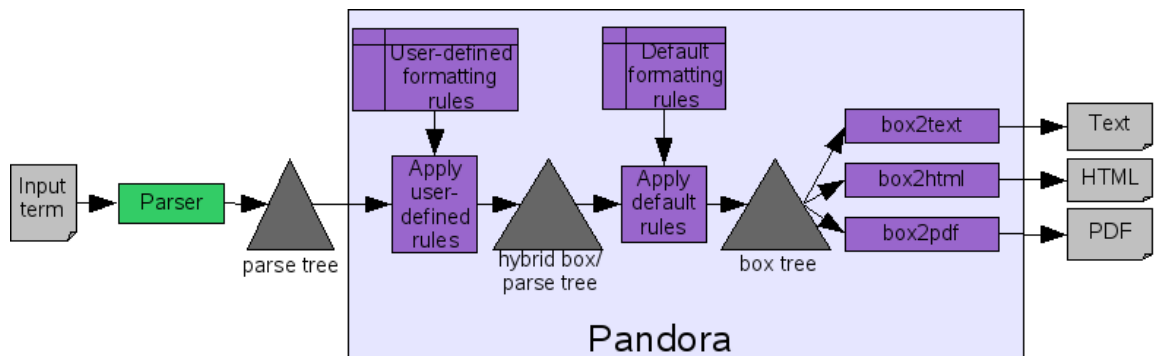
**Table 1.5. Cross-reference Operators (`LBL` and `REF`)**

Operator	Description
<code>LBL</code>	Associate a label with a box
<code>REF</code>	Refer to a labelled box

## Pandora

Recall our general architecture from Figure 1.1, “Formatting Architecture” (page 2) To better understand the formatting process, consider the more detailed architecture in Figure 1.9, “Pandora Formatting Architecture” (page 8).

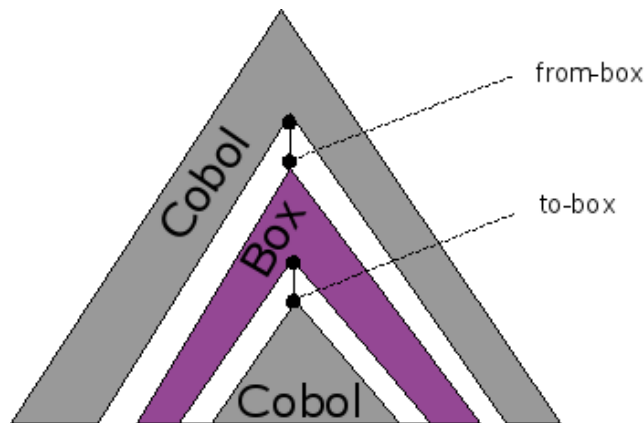
**Figure 1.9. Pandora Formatting Architecture**



The conversion from parse tree to box tree is split in two phases:

- First, user-defined formatting rules are applied to the parse tree. The result is a hybrid box/parse tree that contains box operator wherever user-defined formatting rules have been applied and parse tree operators everywhere else. The transitions between the two are marked with the guards `from-box` (embedding of a box tree in a parse tree) and `to-box` (embedding of a parse tree in a box tree). An example of such a hybrid tree is shown in Figure 1.10, “A Hybrid Box/Parse Tree” (page 8) The parse tree of a Cobol program contains a box tree that in its turn contains a Cobol parse tree.
- Next, the remaining parse tree parts in the hybrid tree are also converted to box expressions according to default rules.

**Figure 1.10. A Hybrid Box/Parse Tree**



The use of hybrid box/parse trees is a unique feature of Pandora that greatly enhances flexibility and generality of formatting.

## Examples

*To be written*

## Historical Notes

The main publications on the Box language and its predecessors are (in historical order) [Cou84], [MCC86], [Vos90], [BV96], [dJ00], and [dJ02].

The main publications on Pandora are [BKVV05] and [BKVV06].

## To Do

- Examples section
- Add links to Historical notes section

## Bibliography

- [Cou84] J. Coutaz. *The box, a layout abstraction for user interface toolkits*. Technical Report CMU-CS-84-167. Carnegie Mellon University. 1984.
- [MCC86] E. Morcos-Chounet and A. Conchon. *PPML: a general formalism to specify pretty printing*. *Information Processing 86*. H.-J. Kugler. Elsevier 1986.
- [Vos90] K. Vos. *For an easy touch of beauty*. Master's thesis. University of Amsterdam. 1990.
- [BV96] M.G.J. van den Brand and E. Visser. *Generation of formatters for context-free languages*. *ACM Transactions on Software Engineering and Methodology*. 5. 1996. 1--41.
- [dJ00] M. de Jonge. *A pretty-printer for every occasion*. Proceedings of the 2nd International Symposium on Constructing Software Engineering Tools (CoSET2000). Wollongong, Australia. . I. Ferguson, Gray J, and L. Scott. June 2000. 68--77.
- [dJ02] M. de Jonge. *Pretty-printing for software engineering*. Proceedings International Conference on Software Maintenance (ICSM 2002). IEEE. . October 2002. 550--559.
- [BKVV05] M.G.J. van den Brand, A.T. Kooiker, J.J. Vinju, and N.P. Veerman. *An Architecture for Context-Sensitive Formatting*. ICSM '05: Proceedings of the 21st IEEE International Conference on Software Maintenance. . 2005. 631--634. IEEE Computer Society
- [BKVV06] M.G.J. van den Brand, A.T. Kooiker, J.J. Vinju, and N.P. Veerman. *A Language Independent Framework for Context-sensitive Formatting*. CSMR '06: Proceedings of the Conference on Software Maintenance and Reengineering. Washington, DC, USA. . 2006. 103--112. IEEE Computer Society Press