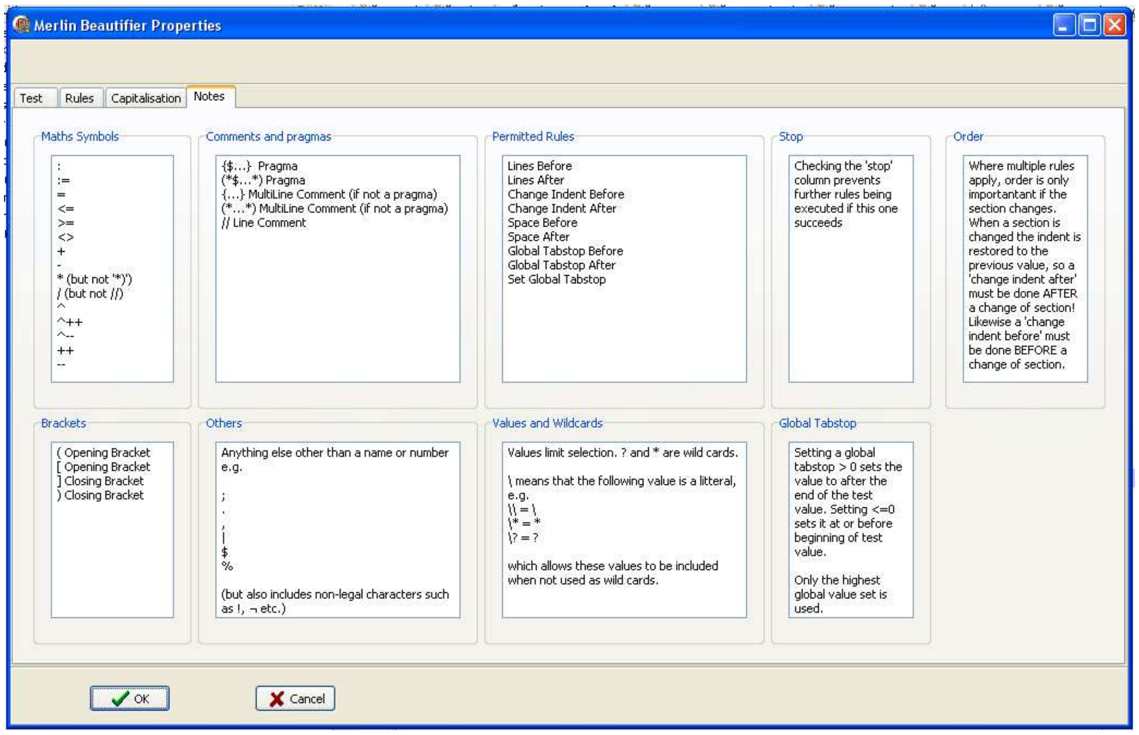
Merlin Beautifier

The Merlin Beautifier is a rule based beautifier. It comprises two parts, a database, which is a series of rules and an engine to execute them.

The beautifier does simple parsing and is not fooled by keyword occurring in, for example, comments or string litterals.

The beautifier can be run in two modes. If no parameters are passed to the program it runs in edit mode. If a valid file is passed to the program is applies the rules in the database. If no parameters are passed the program operates in edit mode allowing the database to be modified.

This document describes how to modify the database, i.e. the program operating in stand-alone mode with no parameters.



Maths Symbols

```
:
:=
=
<=
>=
<>
+
-
* (but not '*)')
/ (but not //)
^
^++
^--
++
--
```

Comments and pragmas

```
{$...}  Pragma
(*$...*) Pragma
{...} MultiLine Comment (if not a pragma)
(*...*) MultiLine Comment (if not a pragma)
// Line Comment
```

Permitted Rules

Lines Before
Lines After
Change Indent Before
Change Indent After
Space Before
Space After
Global Tabstop Before
Global Tabstop After
Set Global Tabstop

Stop

Checking the 'stop' column prevents further rules being executed if this one succeeds

Order

Where multiple rules apply, order is only importantant if the section changes. When a section is changed the indent is restored to the previous value, so a 'change indent after' must be done AFTER a change of section! Likewise a 'change indent before' must be done BEFORE a change of section.

Brackets

```
( Opening Bracket
[ Opening Bracket
] Closing Bracket
) Closing Bracket
```

Others

Anything else other than a name or number e.g.

```
;
.
,
|
$
%
```

(but also includes non-legal characters such as !, ¬ etc.)

Values and Wildcards

Values limit selection. ? and * are wild cards.

\ means that the following value is a litteral, e.g.

```
\\ = \
\* = *
\? = ?
```

which allows these values to be included when not used as wild cards.

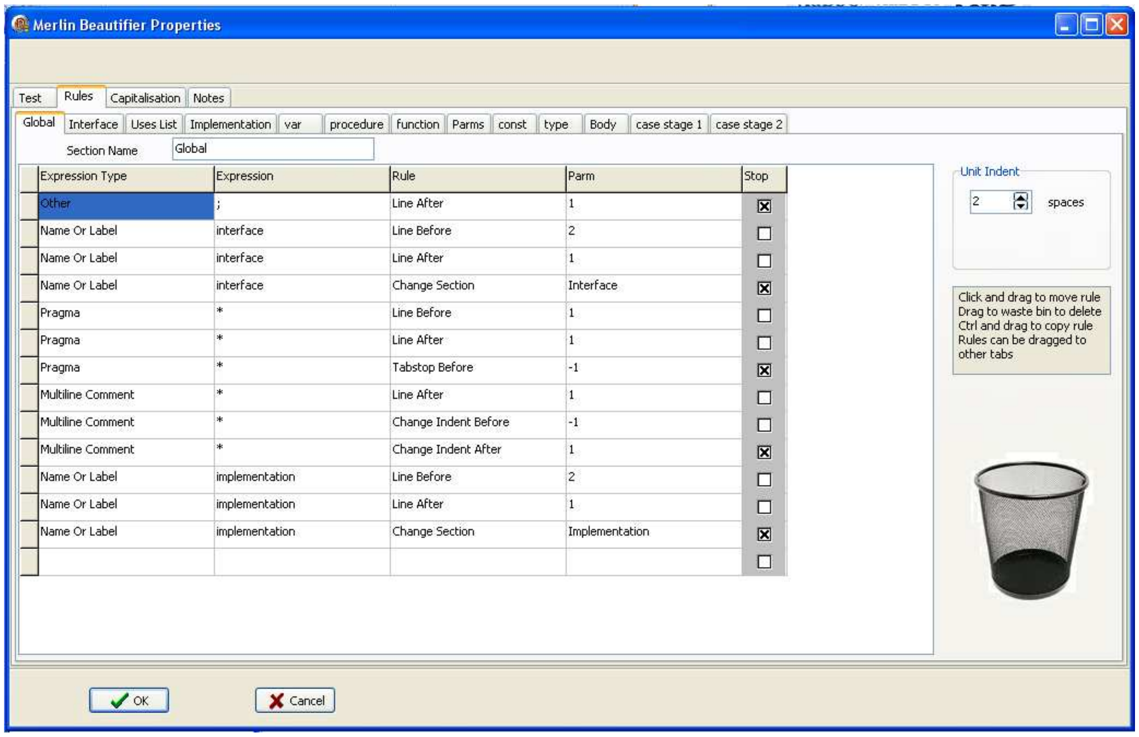
Global Tabstop

Setting a global tabstop > 0 sets the value to after the end of the test value. Setting <=0 sets it at or before beginning of test value.

Only the highest global value set is used.

When the program is started up you see a program with a number of tabs. The above screen shows the Notes page. It has a series of hints on how to use the beautifier.

The database consists of a series of rules.



Section Name: Global

Expression Type	Expression	Rule	Parm	Stop
Other	;	Line After	1	☒
Name Or Label	interface	Line Before	2	☐
Name Or Label	interface	Line After	1	☐
Name Or Label	interface	Change Section	Interface	☒
Pragma	*	Line Before	1	☐
Pragma	*	Line After	1	☐
Pragma	*	Tabstop Before	-1	☒
Multiline Comment	*	Line After	1	☐
Multiline Comment	*	Change Indent Before	-1	☐
Multiline Comment	*	Change Indent After	1	☒
Name Or Label	implementation	Line Before	2	☐
Name Or Label	implementation	Line After	1	☐
Name Or Label	implementation	Change Section	Implementation	☒
				☐

Unit Indent: 2 spaces

Click and drag to move rule
Drag to waste bin to delete
Ctrl and drag to copy rule
Rules can be dragged to other tabs

The rules are separated into sections and subsections. Each section/subsection consists of a series of rules. This method allows different rules to be applied, for instance, to a parameter list and to a list of variables. Rules are applied in the order Current Subsection, Current Section, Global. If a rule is applied and a stop box is checked, further rules are not applied.

Each rule consists of up to 5 parts:

1. Expression Type
   This can be one of the following
   a) None
      Not really a type. If present it does nothing.
   b) NewLine
      A new line. You normally would not use this in a rule, but you can.
   c) Whitespace
      anything like a number of spaces, a tab character and such like. Again, you would not normally use this in a rule, but you can.
   d) Quoted String
      Anything contained within quote marks. These can be single or double quotes. Again, you would not normally use this as a rule. It's main purpose is to prevent accidental formatting of text strings.
   e) Open Bracket
      '(' or '['
   f) Close Bracket
      ')' or ']'
   g) Name Or Label

    Anything that could be construed as a name or number (or label) such as ‘Fred’, 123, and so on. Even illegal values like 12Bill are considered to be name or label.

   h) Maths Symbol
      Things like +, -, *, /, :=, = and so on. It does *not* include textual operators lie ‘and’, ‘div’ ‘xor’ and so on, which are considered to be name or label.
   i) Line Comment
      anything starting //
   j) Multiline Comment
      anything contained in { } or (* *)
   k) Pragma
      a multiline comment whose first character is $, e.g. {$W+}
   l) Other
      Anything else, like ‘:’, ‘;’, ‘,’ and so on.
2. Expression
   The literal value that must be matched for the rule to apply. Wild cards can be used. ‘?’ allows any value at that position, but requires there to be something there. ‘*’ allows any set of values. For example ‘end*if’ would allow ‘end_if’ or ‘endif’, but unfortunately would also allow ‘ending_if’ and loads of other stuff, so be careful. If you need a special value like ‘*’ in your test prefix it with ‘\’.
3. Rule
   This is what do if the rule Expression and expression type match. The following are possible
   a) None
      Does nothing. Can be used with stop to prevent further rules being applied.
   b) Space Before
      Ensures that at least the number of spaces specified in the parm column are placed before the current expression.
   c) Space After
      Ensures that at least the number of spaces specified in the parm column are placed after the current expression.
   d) Line Before
      Ensures that at least the number of lines specified in the parm column are placed before the current expression.
   e) Line After
      Ensures that at least the number of lines specified in the parm column are placed after the current expression.
   f) Change Indent Before
      Changes the indent specified by the value specified in the parm column multiplied by the unit indent specified to the right (above the waste paper bin). This is applied to the current expression. Indents are saved and restored when sections and subsections are changed, so be very careful that if an expression both changes an indent and changes a section that the Change Indent Before comes **before** change subsections rule. The value in the parm field will normally be -1.
   g) Change Indent After
      Changes the indent specified by the value specified in the parm column multiplied by the unit indent specified to the right (above the waste paper bin). This is *not* applied to the

current expression. Indents are saved and restored when sections and subsections are changed, so be very careful that if an expression both changes an indent and changes a section that the Change Indent Before comes **after** change subsections rule. The value in the parm field will normally be 1.

h) New Indent Before
This sets the indent specified by the value specified in the parm column multiplied by the unit indent specified to the right (above the waste paper bin). This is applied to the current expression. Indents are saved and restored when sections and subsections are changed, so be very careful that if an expression both changes an indent and changes a section that the Change Indent Before comes **before** change subsections rule.

i) New Indent After
This sets the indent specified by the value specified in the parm column multiplied by the unit indent specified to the right (above the waste paper bin). This is *not* applied to the current expression. Indents are saved and restored when sections and subsections are changed, so be very careful that if an expression both changes an indent and changes a section that the Change Indent Before comes **after** change subsections rule. The value in the parm field will normally be 1.

j) Tabstop Before, Tabstop After
Tabstops are every 8 characters. Tabstop n means attempt to line up at position 8*n. If it is not possible then line up to the next available tabstop. A value of -1 means go to the left margin.

k) Change Section
A section is a partition of the program, for example interface or implementation. Different rules can apply to different sections. These are top level partitions, so you an only change from one section to another. The subsection is entered in the Parm column. ‘<new>’ can be used to add a new section or subsection.

l) Enter Subsection
Subsections can be nested, like subroutines, although only the current active subsection is tested for rules (not the entire stack of subsections). Use Enter Subsection to push a new subsection on to the stack and make it the active subsection.

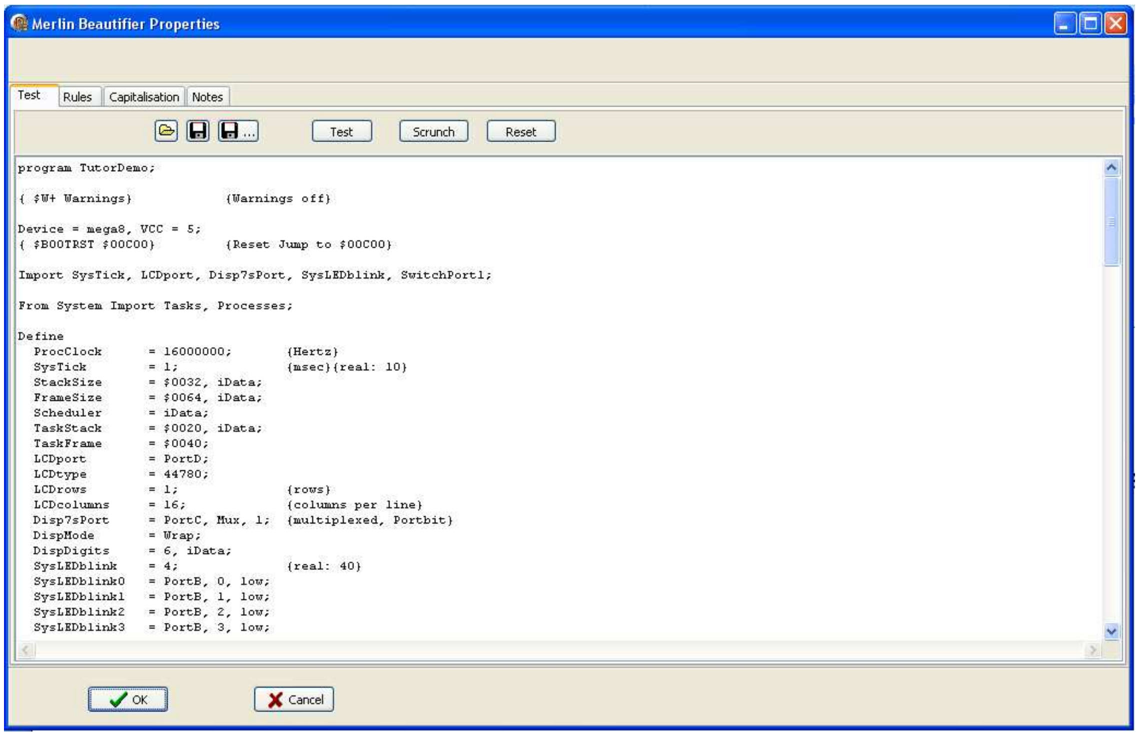
m) Change Subsection
Replaces the current subsection with a new one.

n) Exit Subsection
Leave the current subsection (and ‘pop’ the previous one, if any). The indentation active when the subsection was entered is reinstated. See notes on ‘Change Indent Before’ and ‘Change Indent After’. Failure to observe these rules will lead to unexpected results.

4. Parm
A modifier for the rule. See previous section.
5. Stop
Prevents further rules from being executed if this one succeeds.

Testing

A number of tools exist to help with testing. Files can be loaded, saved, scrunched, tested and viewed.



```
program TutorDemo;

{ $W+ Warnings}           {Warnings off}

Device = mega8, VCC = 5;
{ $BOOTRST $00C00}        {Reset Jump to $00C00}

Import SysTick, LCDport, Disp7sPort, SysLEDblink, SwitchPort1;

From System Import Tasks, Processes;

Define
  ProcClock      = 16000000;       {Hertz}
  SysTick        = 1;              {msec}{real: 10}
  StackSize      = $0032, iData;
  FrameSize      = $0064, iData;
  Scheduler      = iData;
  TaskStack      = $0020, iData;
  TaskFrame      = $0040;
  LCDport        = PortD;
  LCDtype        = 44780;
  LCDrows        = 1;              {rows}
  LCDcolumns     = 16;             {columns per line}
  Disp7sPort     = PortC, Mux, 1;  {multiplexed, Portbit}
  DispMode       = Wrap;
  DispDigits     = 6, iData;
  SysLEDblink    = 4;              {real: 40}
  SysLEDblink0   = PortB, 0, low;
  SysLEDblink1   = PortB, 1, low;
  SysLEDblink2   = PortB, 2, low;
  SysLEDblink3   = PortB, 3, low;
```

loads a file. Once it is loaded it is remembered, so there is no need to load the file each time.

saves a file.

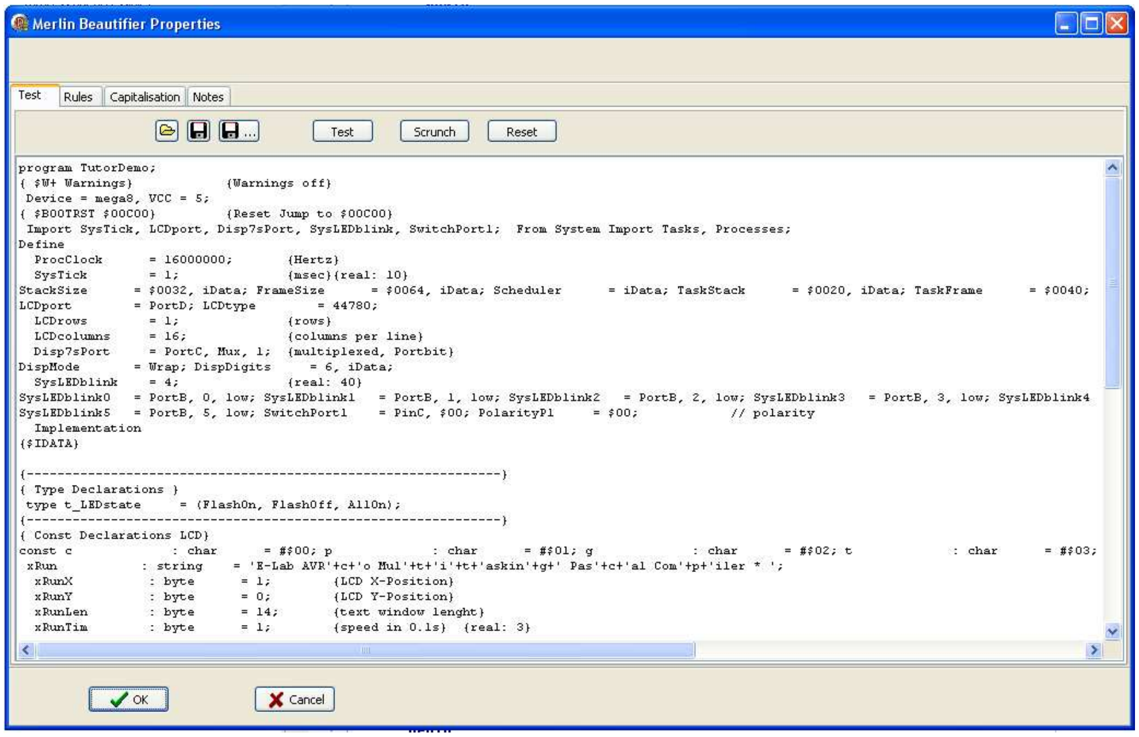
… saves a file with a different name.

Reset restores a file to its pristine, unmodified state.

Scrunch 'scrunches' a file, removing lots of spaces without making it unusable (just pretty unreadable).

Test tests the current rules.

Here is a scrunched example:

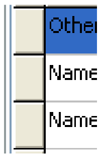


```
program TutorDemo;
{ $W+ Warnings}           {Warnings off}
 Device = mega8, VCC = 5;
{ $BOOTRST $00C00}        {Reset Jump to $00C00}
 Import SysTick, LCDport, Disp7sPort, SysLEDblink, SwitchPort1;  From System Import Tasks, Processes;
Define
  ProcClock      = 16000000;       {Hertz}
  SysTick        = 1;              {msec}{real: 10}
StackSize      = $0032, iData; FrameSize      = $0064, iData; Scheduler      = iData; TaskStack      = $0020, iData; TaskFrame      = $0040;
LCDport        = PortD; LCDtype        = 44780;
  LCDrows        = 1;              {rows}
  LCDcolumns     = 16;             {columns per line}
  Disp7sPort     = PortC, Mux, 1;  {multiplexed, Portbit}
DispMode       = Wrap; DispDigits     = 6, iData;
  SysLEDblink    = 4;              {real: 40}
SysLEDblink0   = PortB, 0, low; SysLEDblink1   = PortB, 1, low; SysLEDblink2   = PortB, 2, low; SysLEDblink3   = PortB, 3, low; SysLEDblink4
SysLEDblink5   = PortB, 5, low; SwitchPort1    = PinC, $00; PolarityP1     = $00;            // polarity
  Implementation
{$IDATA}

{--------------------------------------------------------------}
{ Type Declarations }
 type t_LEDstate     = (FlashOn, FlashOff, AllOn);
{--------------------------------------------------------------}
{ Const Declarations LCD}
const c            : char      = #$00; p            : char      = #$01; g            : char      = #$02; t            : char      = #$03;
 xRun           : string    = 'E-Lab AVR'+c+'o Mul'+t+'i'+t+'askin'+g+' Pas'+c+'al Com'+p+'iler * ';
  xRunX          : byte      = 1;        {LCD X-Position}
  xRunY          : byte      = 0;        {LCD Y-Position}
  xRunLen        : byte      = 14;       {text window lenght}
  xRunTim        : byte      = 1;        {speed in 0.1s}  {real: 3}
```

Drag and Drop

You can grab a rule by dragging the far left hand button



(looks like a grey block). A rule can be dragged to another position (even on another tab) or to the rubbish bin (to delete it).