GNU enscript

For version 1.6.3, 14 January 1999

Markku Rossi

This is the first edition of the GNU enscript documentation,
and is consistent with GNU enscript 1.6.3.

Short Contents

Table of Contents

1 Introduction

- overall
- design

2 Invoking Enscript

3 Basic Printing

3.1 Input Encodings

3.2 Selecting Fonts

3.3 Page Headers

3.4 Page Handling

3.4.1 Page Orientation

3.4.2 N-up Printing

3.4.3 Fitting Text to Page

3.5 Highlighting

3.5.1 Different Output Languages

4 Advanced Usage

4.1 Selecting Pages

4.2 Escape Sequences

4.3 Input Filters

4.4 Slice Printing

4.5 PostScript Printer Controlling

4.6 Pass-Through Mode

5 Configuration Files

6 Customization

6.1 Output Media

6.2 User-Defined Fancy Headers

7 The ‘states’ Program

8 Writing New Highlighting Definitions

The highlighting works in three separate phases. First, the *highlighing rules* process the input stream and parse it into logical components. The components are called *faces*. A face presents one logical component of the input language, for example, a keyword, a comment, etc.. The enscript's highlighting model defines the following faces:

bold
italic
bold_italic Hard-coded faces for the bold, italic, and bold-italice text types. These faces define the exact presentation of the face font, so the style files have very little power in customizing their outlook. These faces should be avoided as much as possible.

comment A comment, normally in a programming language.

function_name
A function name. The function names are normally recognized from function definitions, not from an use of the function.

variable_name
A variable name. The variable names are normally recognized from function, type, and variable definitions.

keyword A reserved keyword. Normally, all occurrences of the keywords are recognized.

reference A reference to another location in a file or to another file or resource. For example, in the C-language, the goto targets are references.

string A string literal.

builtin A builtin function or property. Normally, all occurrences of the builtins are recognized.

type A type specifier. The types are normally recognized from function, type, and variable definitions.

As the second step, the *output style* specifies how the faces are presented in the generated output. Each face has the following properties:

fontname The PostScript font name of the the font that is used for the face. This property is used only for the PostScript outputs.

boldp A boolean flag which tells whether the face should be printed in bold font. This property is used for all output languages except for the PostScript which uses the fontname property.

italicp A boolean flag which tells whether the face shuold be printed with italic font. This property is used for all output languages except for the PostScript which uses the fontname property.

fg_color The foreground color of the face.

bg_color The background color of the face. This property is not implemented on all output languages.

Finally, the *output language* describes how the faces and other text are presented in the output language. The output language defines a set of functions which are called to generate the output.

8.1 Highlighting Rules

8.2 Styles

8.3 Output Languages

map_color (*r, g, b*) Function
language_print (*string*) Function
language_symbol (*symbol*) Function
header () Function
trailer () Function
face_on (*face*) Function
face_off (*face*) Function
LANGUAGE_SPECIALS Variable

The following example creates a new output language `simple_html` that creates simple HTML outputs. The output language is defined in a file called '`lang_simple_html.st`'. The file must define a state called `lang_simple_html`. The file can be located in any directory that is in the load path of the states program.

The output language definitions are defined in the `BEGIN` block of the `lang_simple_html` state. Please, note that the `BEGIN` block is ended with a `return`-statement. This statement will return the control to the calling state that is the start state of the enscript highlight program. If the `return`-statement was omitted, the states would start processing the input with the `lang_simple_html` state which is obviously a wrong choice.

```
state lang_simple_html
{
  BEGIN {
    sub map_color (r, g, b)
    {
      return sprintf ("#%02X%02X%02X", r, g, b);
    }

    sub language_print (str)
    {
      str = regsuball (str, /\&/, "&amp;");
      str = regsuball (str, /</, "&lt;");
      str = regsuball (str, />/, "&gt;");
      str = regsuball (str, /\"/, "&quot;");
      print (str);
    }

    sub language_symbol (symbol)
    {
      return false;
    }

    sub header ()
    {
      print ("<html>\n<head>\n<title>Simple HTML Output</title>\n");
      print ("</head>\n<body>\n");
    }

    sub trailer ()
    {
      print ("</body>\n</html>\n");
    }

    sub fase_on (face)
    {
      if (face(boldp])
        print ("<B>");
      if (face(italicp])
```

```
        print ("<I>");
      if (face[fg_color])
        print ("<FONT COLOR=\", face[fg_color], "\">");
    }

    sub face_off (face)
    {
      if (face[fg_color])
        print ("</FONT>");
      if (face[italicp])
        print ("</I>");
      if (face[boldp])
        print ("</B>");
    }

    LANGUAGE_SPECIALS = /[<>\&\"]/;

    return;
  }
}
```

Index