Writing Printer and Device Definition Databases

Chapter 1

Overview

The printer and device definition databases are ASCII text files that describe the printer capabalites. The text file is compiled into a binary format using the ddigest" command. The primary purpose of compiling the text file is to minimize any problems that may be caused in the future such as when running the markvision utility or at print time, due to problems in syntax or logical description of the printer options and menus flow.

The PDDs serve the following two functions:

Information base for the user interface

The PDD files provide all the text for displaying while prompting for selection of queue data stream or printer capabilities. While doing queue management option 5 thru 8 in the lexprt utility, the information and the flow control of the user interface is driven by the PDD file starting at the screen for dData Stream Options". All the information on the following screens comes from the PDD files.

Database for Printer Codes

The PDD files provide all the printer codes for all the supported options and their management to be sent to the printer at print time. The dformatter" program is called at print time by the printer sub-system backend, which reads the virtual queue definition file created by the user-interface when managing the queue, i.e. creating /

changing. The formatter than refrences the compiled version of the PDD in the /etc/marknet/etc/pdd/bin directory to obtain the printer codes for that printer and that queue.

Rules for writing PDDs and DDDs

There are many rules applied when writing the PDDs. A certain grammer has been used to define the printer defintion databases. The following section describes these rules using examples.

These are the data types recognized by the compiler

```
KEYWORDS        STRING            INTEGER
```

String

Strings are defined as a sequence of ASCII characters enclosed by double-quotes (""). Two consecutive double-quotes are invalid. For example

```
label ""    /* <-- Invalid  */

label "  "  /* <-- Valid    */
```

The newline character '\n' in a string will produce syntax errors. A STRING should NOT have any new line characters. In some fields the word "none" is allowed to point out an empty field in the pdd_block.

Integer

INTEGERs are defined as a series of numerics (integer). Floats or any other non-integer numeric data is not supported.

Keywords

There are a number of words which are considered as keywords by the compiler. The following is a list of these words :

```
banner_init_sequence        ddd_block
ddd_file                    decimal
default_value               desc
end_string                  help
init_modes                  init_sequence
item                        label
list                        max
max_length                  menus
next_ptr                    number
option_type                 option_value
p_code                      pdd_block
pdd_file                    prompt
special_char1               special_char2
special_char3               special_string1
special_string2             special_string3
string                      sub_list
sub_menu                    sub_number
sub_string                  tag
title                       transport
valid_type
```


Block Types

There are a few "block types" defined for the PDD files. These types are descibed as the logical blocks for the file. These types are:

```
PDD_BLOCK                   LIST
STRING                      NUMBER
MENU
```

Block Headings

All blocks (pdd_block, list, number, string, menu) have a common header section that describes the block as follows:

Line #1 thru #3 define the block header. The *"title"* and the *"prompt"* fields describe the field for use by the user-interface. The *"help"* field provides a help tag

PDD_BLOCK

The "pdd_block" defines the first few blocks in the PDD file. This block provides the information needed by the "formatter" program at print time. It has the following fields:

```
init_sequeuce              banner_init_sequence
init_modes                 end_string
special_string1            special_string2
special_string3            special_char1
special_char2              special_char3
```

init_modes

Some printers require an initialization sequence to set it in a certain emulation mode. The STRING field that follows this keyword contains that initialization sequence. The word "none" sends NO initialization sequence. This field is the FIRST one the "formatter" processess if active.

An Exmaple of an active field:

```
init_modes "${27}%-12345X@PJL ENTER LANGUAGE =PCL"
```

To disable sending any codes for printer initialization :

```
init_modes  "none"
```

init_sequence

This field is of type STRING (refere to the section on data types). This field sets up the printer codes and sends them before the data. It has the list of all tags defined later in the file. For example, in the printer defintion file for the IBM_LaserPrinter_4039_16L this field is as follows:

```
FORMAT:  init_sequence  STRING
```

Example:

```
init_sequence="pcl_resolution,pcl_crlf,
      pcl_text_wrap,pcl_duplex,
      pcl_paper_size,pcl_paper_tray,
      pcl_orientation,pcl_symbol_set,pcl_font,
      pcl_pl66,pcl_top_margin,
      pcl_page_length,pcl_indentation,
      pcl_page_width"
```

The above is the sequence in which the printer options will be processed by the formatter. This STRING is a "comma" separated list of printer options for the data strem "pcl". The idea of using the pre-fixed name"pcl" with every option is to make the name uniq for every data strem. The formatter will run a loop at print time and will fetch the appropriate printer codes from the PDD binary file first for the "pcl_resolution", than for the "pcl_crlf" option and so on. Therefor, if for some reason you want the ignore a certain option at print time ONLY, take that option out of this list to disable sending the related codes for that option to the printer at print time.

A good example, of the above description is the fonts in PCL5 on all the supported printers. The output of the font depends on the point size or pitch selected.

For example, the following is the "p_code" for a Courier font:

```
p_code "${27}(s0p$${pcl_pitch}h12v0s0b3T"
```

This example shows the "p_code" for CG Times font:

```
p_code "${27}(s1p$${pcl_point_size}v0s0b4101T"
```

The dependency of the font on point size or pitch can be seen from the above example. The impact of this dependency is the option for point size and pitch is displayed for user selection in the user interface, but when printing only the "pcl_font" option is part of the "init_sequence" above. There is no "pcl_point_size" or "pcl_pitch" field in the "init_sequence" field. The fact that the value for the pcl_point_size / or pcl_pitch is a part of the pcl_font, we don't need these two options as a part of the "init_sequence".

banner_init_sequence

This field is of type STRING. This field sets up the printer codes and sends them before the banner page (if enabled). It has the list of all tags defined later in the file. The field controls the options for the banner page.

For example, assume the following scenario of providing control for the following three options for the banner page: pcl_banner_tray, pcl_banner_resolution, pcl_banner_paper_size.

The "banner_init_sequece" field should be as follows:

```
banner_init_sequece  "pcl_banner_resolution,
pcl_banner_tray, pcl_banner_paper_size"
```

The above mentioned three (3) fields should be defined in the file. From the user interface point of view the "menu" block which will be discussed later in more detail, should look as follows:

```
menu "pcl_banner_options" {
        title           "PCL Banner Options"
        prompt          "PCL Banner Options"
        help            "PCL_Banner_Options_Menu_Help"
        next_ptr        "install_queue()"

        sub_list pcl_banner_page <--- line #1 sub_list
        pcl_execute_banner      <--- line #2
              sub_list pcl_banner_tray      <--- line #3
              sub_list pcl_banner_resolution<--- line #4
              sub_list pcl_banner_paper_size<--- line #5
}
```

Line #3 through line #5 represent the options specified in the "banner_init_sequece" field. Line #1 and line #2 are special tags which control the printing of the banner page.

<data_stream>_banner_page

Enables or disables printing of the banner page. If it is disabled (i.e == no) the "formatter" program ignores printing the banner page and any related options. If this is enabled (i.e == yes) the "formatter" will print the banner page and will process all the options in the "banner_init_sequence" field.

<data_stream>_execute_banner

This tag controls handling of how the banner page program is used. By default, the banner program is an executable, which will take five (5) arguments in the following sequence

```
banner_program <title> <user> <host> <queue>
<printer_des> <paper- siz<letter_size>
```

Normally the banner program defined by the

<data_stream>_banner_file

The *"formatter"* will execute that file if it has execute permissions set and this tag (i.e <data_stream>_execute_banner set to "yes"). If this tag is set to "no", the formatter will print (i.e. cat) the banner_file instead of running it. This way we have the option of either running the banner program with the supplied options or simply sending the file to the printer.

The compiler will check at compile time for the existance of the tags associated with the pdd_block later as one of the options in the first (special case) list block. For example, for the IBM LaserPrinter 4039 PDD file, the pdd_blocks are as follows:

```
pdd_block "pcl" {    <----- line #1
          .
          .
          .
}

pdd_block "ps" {     <------ line #2
          .
          .
```

```
        .
}

list    "ds_list" {

        title           "Data Stream Options"
        prompt          "Data Stream Options"
        help            "DS_LIST_Help"

        option_type list {

              label           "PCL Data Stream"
              desc            " "
              value           "pcl"   <-------- line #3
              next_ptr        "pcl_options"
              label           "PS Data Stream"
              desc            " "
}
```

As we can see that line #1 and #3 are describing "pcl", similarly line #2 and #4 are describing "ps". If the "value" STRING value STRING (e.g. value "pcl" was not defined as the tag for one of the pdd_blocks above in the begining of the file, an error message is displayed as :

```
Checking syntax ...

63: Missing predefined Initialization Block at
"pcl_options"

Compilation aborted.
```

LIST Block

The "list" block is defines a set of pre-defined valid options for any printer emulation. For example, in PCL5 emulation on the *IBM LaserPrinter 4049 (Optra family)* "Resolution has 3 resolution options 300 dpi, 600 dpi and 1200 dpi.

Therefore, the "Resolution" would be a perfect option to be setup as a *"list"* block. In other words, *"list"* block is used for options that have a LIST of options the user can choose from. When the user selects this option, another list of options is displayed for selection. There is always one option, which is the default item. In order to change the default item in the list the *"default_item*" keyword should be moved to the top of the new option as follows:

For PCL orientation, the following example shows the default item to be *"Portrait"*

```
list    "pcl_orientation" {

        title           "Orientation"
        prompt          "Orientation"
        help            "PCL_Orientation_Help"

        option_type list {

               default_item      <---- line #1
               label            "Portrait"
               desc             "Set orientation to portrait"
               value            "portrait"
               p_code           "${27}&l0O"

               label           "Landscape"
               desc            "Set orientation to landscape"
               value           "landscape"
               p_code          "${27}&l1O"

               .
               .
               .
        }
```

```
}
```

Now to change the default item to be "Landscape" move line #1 (default_item) to the next option as follows:

```
list    "pcl_orientation" {<--- line #2
        title           "Orientation"
        prompt          "Orientation"
        help            "PCL_Orientation_Help"

        option_type list {

               label           "Portrait"
               desc            "Set orientation to portrait"
               value           "portrait"
               p_code          "${27}&l0O"

               default_item            <---- line #1 (new)
               label           "Landscape"
               desc            "Set orientation to landscape"
               value           "landscape"
               p_code          "${27}&l1O"

               .
               .
               .
        }
}
```

Once a change is made to a printer defintion file, it should be re-compiled. Refer to the section on re-compiling PDDs.

The STRING following the keyword "list" as in line #2 is the tag name and should be uniq in the PDD file. The "label" and the "desc" keywords are followed by a string for displaying by the user-interface. The "value" field, is the identifier which is placed in the virtual queue definition (vqd) file when a queue is created. For exmaple, if the user selects "Landscape" for orientation in pcl for the 4049 printer, the following line is placed in the vqd file:

```
pcl_orientation=landscape
```

The same is done for all LIST block tags. The vqd file is text file that describes the queue setup with tag=value pairs for all supported options.

The "p_code" stands for the printer code, which is actually sent to the printer for that option. It is a STRING, which has a special format. This string is parsed at print time, by the "formatter" program. There are two important translations done while sending this string to the printer.

Any number from 0 to 255 starting with '${' and ending with '}' is translated to its decimal equivalent before sending to the printer. Therefore, in order to send an ESCAPE we place ${27}. To send the equivalent of decimal 10 (new line), we place ${10}.

Refer to the ascii man page for a list of all characters.

An example of this translation is as follows:

```
p_code "${27}F3n${10}${12}${13}"
        ^^^^    ^^^^ ^^^^ ^^^^
         |       |    |    |
         |       |    |    |
        ESC      NL   NP   CR
```

If a printer code can use a variable numeric field as a part of the printer code STRING, we delimit that part with '$${' as the begining delimiter, and '}' as the ending delimiter. For example, in PCL5 for "indentation"

EXAMPLE :

At print time, anything inside the $${ .... } is processed so that it is evaluated and variable substitution is done. Therefore, in the above example, say pcl_indentation was equal to 5 than the sequense sent to the printer will be

```
STRING   "${27}&a$${pcl_indentation}L"
DECIMAL  27 38 97 53 76
HEX      1B 26 61 35 4C
```

Here is the translation:

STRING	DECIMAL	HEX
${27}	27	1B
&	38	26
a	97	61
$${pcl_indentation}	53	35
L	76	4C

The sequence in between '$${' and '}' is evaluated. Therefore, any complex mathematical expression can be substituted in between these delimiter fields. For example, say the page width is dependent on more than one variable is calculated on the fly :

EXAMPLE 1:

Take the example of the p_code field for pcl_page_width

```
p_code "${27}&a$${pcl_page_width + pcl_indentation -1}M"
if pcl_page_width = 70
   pcl_indentation = 5
```

the expression evaluates as:

```
pcl_page_width + pcl_indentation -1 = 70 + 5 -1 = 74
```

Therefore the hexadecimal string sent to the printer is :

```
HEX      1B 26 61 37 34 4D
DECIMAL  27 38 97 55 52 77
```

More complex calculations can be done:

```
p_code "${27}&a$${(((pcl_page_width -1 )/3)+(pcl_indentation*4))}"

         if pcl_page_width = 70
            pcl_indentation = 5
(((pcl_page_width -1 )/3)+(pcl_indentation*4)) =
   ((( 70 -1 )/3)+(5*4)) = (( 69 /3 ) + ( 20 ))= ( 23 + 20 )= 53
```

Therefore:

```
HEX       35  33

DECIMAL   53  51
```

is sent to the printer. All rules of operator precedence are applied.

Order of LIST block fields:

The syntax of the LIST block (and all other blocks) is very rigid. All the field should be in the same sequence. The only field which is interchangable is the "default_item" field. It can be placed on the TOP of any option to make it the default item by default.

The LIST block can appear inside a MENU block (discussed later), or it can be the first LIST block. The "p_code" field and "next_ptr" field can be used as appropriate. The fact that the first LIST block is a special list block and the tag is "ds_list". This tag should always be the tag for the first LIST block in a PDD. The user interface displays this first LIST block as the starting point. This first LIST block displays the available emulations options and upon user selection will point to the appropriate printer options menu using the "next_ptr" field. For example, the following is an example of the 4039 PDD:

```
list    "ds_list" {

         title           "Data Stream Options"
         prompt          "Data Stream Options"
         help            "DS_LIST_Help"

         option_type list {

               label           "PCL Data Stream"
```

```
        desc            " "
        value           "pcl"
        next_ptr        "pcl_options"<--- line #1

        label           "PS Data Stream"
        desc            " "
        value           "ps"
        next_ptr        "ps_options"<--- line #2

        default_item
        label           "AUTO"
        desc           "(select auto sniffing data stream)"
        next_ptr        "auto_options"<--- line #3

        label           "Passthru"
        desc            "(send raw data to the printer)"
        value           "passthrough"
        next_ptr        "install_queue()"<--- line #4
    }
}
```

Line #1: uses the "next_ptr" field to point to the NODE with tag name of "pcl_options".

Line #2: uses the "next_ptr" field to point to the NODE with tag name of "ps_options".

Line #4: uses the "next_ptr" field to point to the FUNCTION with the name "install_queue()"

The compiler in the first pass creates an internal table of all the items associated with the "next_ptr" field. In the second pass the items in this list are checked against all nodes (list, string, number and menu). If any "next_ptr" is missing or not defined in the PDD an error message is returned.

The other LIST block fields should maintain the same format and sequence as show above, i.e

```
default_item                    (optional)
label               STRING      (mandatory)
desc                STRING      (mandatory)
```

```
value                STRING      (mandatory)
next_ptr             STRING      ----
p_code               STRING      ----
```

STRING Block

STRING blocks are used for options that require user input in terms of an ascii characters. For pure numeric inputs NUMBER nodes / blocks should be used. For the sake of explanation, "nodes" and "blocks" are used interchangebly in this document. The following is the syntax for STRING blocks:

```
string "pcl_banner_file" {

        title           "Banner filename"
        prompt          "Absolute Path for the Banner File"
        help            "PCL_Banner_File_Help"

        option_type  string {

                    valid_type              1 2 3
                    default_string          "banner_pcl"
                    exclude_chars_set       ";!"
                    include_chars_set       "/.-_,"
                    max_length              255
                    validation_function     "valid_reg_path()"
                    p_code                  "none"
        }
}
```

The keyword *"string"* at the begining of the block is the node type. The string that follows i.e. *"pcl_banner_file"* is the node name / tag name for the node. This should be a uniq node name.

The header block that comprises of "*title*", "*prompt*" and "*help*" defines the node. The node / block properties are defined in the section:

```
option_type  STRING {
      ....
      ...
}
```

The following sections describes these fields.

valid_type

This field defines the type of characters allowed for this node. The numbers following this field are actually pointers to pre-defines character classes. These numbers are ORed by the compiler and interpreted accordingly by the user-interface. The following table describes these masks:

Number	Mask	Description
0	NONE	No charcter is valid
1	DIGIT	Only numeric digits are valid.
2	ALPHA	Only alphabets are valid.
3	ALNUM	Only numerics and alphabets are valid.
4	SPACE	Only space and tabs are valid.
8	PUNCT	Only punctuations are valid.
16	CNTRL	Only Control characters are valid.
31	ALL	All characters are valid.

STRING node fields

The above table describes the diffrent charcters that can be allowed. The compiler internally OR these numbers to get the valid super set. For example, if we want to allow DIGITs and PUNCT, the valid_type field should be as:

```
valid_type   8
```

The compiler will perform (1 OR 8 = 9). This value is stored in the binary file. Different combinations of these numbers can be placed to allow a certain character set.

`default_strings  STRING`

This field holds the default string for this tag.

`exclude_chars_set STRING`

This field holds the list of characters to be excluded. This gives the PDD better control for the inclusion of all possible characters using the *"valid_type"* field and then excluding a certain number of

`include_chars_set STRING`

This field holds the list of characters to be included. This gives the PDD better control for deciding which characters to include form the list of excluded characters.

`max_lengthNUMBER`

This field holds the maximum length of this ascii string.

`validation_function  STRING`

This field holds the name of the function to be used by the user interface to validate the string entered by the user. If the name of the function is *"validate_file_path"*, the string should read:

`validation_function  "validate_file_path()"`

The `"()"` round braces are important and should be there for the user interface to locate the correct function.

`p_code STRING`

This is a reserved field. It is not used currently for string tags. It should be placed in this location with STRING equals to *"none"*.

NUMBER Blocks

NUMBER blocks are used for options that require user input in terms of a numbers. The following is the syntax for NUMBER blocks:

```
number "ppds_top_margin" {
              label   "Top Margin"
              prompt         "Enter Top Margin :"
              help        "PPDS_Top_Margin_Help"

              option_type number {

                    default_value    5
                    decimal          0
                    min              0
                    max             60
                    number_type      1
                    validation_function   "test_range()"
                    p_code  "${27}&l$${ppds_top_margin}E"
              }
}
```

The keyword *"number"* at the begining of the block is the node type. The string that follows i.e. *"ppds_top_margin"* is the node name / tag name for the node. This should be a uniq node name. The header block that comprises of *"title"*, *"prompt"* and *"help"* defines the node. The node / block properties are defined in the

```
"option_type" STRING { .... }
```

section. The following sections describes these fields:

Description of NUMBER node fields

default_value NUMBER

This field holds the default value for this tag. This will be the value substituted by the user interface to the virtual queue definition file.

`decimal  NUMBER`

This field holds the number of decimal places that are allowed for this tag if any. If the field is an integer, this will be "0". For one decimal place it will be set to 1 and so on.

`min  NUMBER`

This field holds the minimum range for this field. This is used by the user interface to validate the range when a number is entered by the user.

`max NUMBER`

This field holds the maximum range for this field. This is used by the user interface to validate the range when a number is entered by the user.

`number_type  NUMBER`

This field holds the type of integer. Different data streams treat these numbers diffrently. Conisder the following two examples for two diffrent data streams.

PCL

For example, to set the page length to 60 lines per page the following command is sent to the printer:

```
Format    ESC &  l <page_width> F

ASCII     ESC &  l    6    0    F

DECIMAL    27   38  108   54   48   70

HEX        1B   26   6C   36   30   46
```

This shows that for a page length of 60 lines per page we send the ascii characters "60" i.e. 0x36 0x30.

For example, to set the page length to 60 lines per page the following command is sent to the printer:

```
Format     ESC     C   <lines>

DECIMAL     27    67   60

HEX         1B    43   3C
```

This shows that for a page length of 60 lines per page we send the the byte value of 60 which is hex 3C.

This situation creates a distinction of how the "formatter" program is going to handle these number fields. In order for the formatter to know the difference, this field is used to flag the format of these numbers (ASCII_TYPE of VALUE_TYPE). Currently only two formats are supported. This can be extended to other types if needed in the future. The following table describes the currently supported values for this field.

Number	Representation	Description
0	ASCII_TYPE	This type is used to send characters with their ascii representation as in PCL5
1	VALUE_TYPE	This type is used to send integer values as in the PPDS example

`validation_functionSTRING`

This field holds the name of the function to be used by the user interface to validate the number entered by the user. If the name of the function is *"test_range"*, the string should read:

```
validation_function "test_range()"
```

The "()" round braces are important and should be there for the user interface to locate the correct function.

p_code STRING

This field holds the printer code for this tag. This STRING will normally conatain the codes along with the tag surrounded by $${ and }. For example, in PCL5 for page length:

```
p_code "ESC&l$${page_length}F"
```

The *"formatter"* will replace the '$${page_length}' with the value found for page_length in the virtual queue definition for the queue in question. For example, if the VQD file is as follows:

```
.
.
.
pcl_indentation=5
pcl_page_length=40
pcl_banner_page=yes
.
.
.
```

If "number_type 0" is set p_code is substituted as follows :

```
ESC &  l   4   0  F

27  38 108 52  48 70
```

If "number_type 1" is set p_code is substituted as follows

```
ESC &  l   (  F

27  38 108 40 70
```

MENU Block

The menu block defines the combination section for all the above refrenced types (i.e. LIST, STRING and NUMBER). Menus can be nested to 10 levels. Nesting here, implies that sub_menus can be used as an item within a menu and up to 10 levels deep.

For example, the automatic sniffing queue is a superset of the PCL and PostScript menu blocks. The following section describes the format of the menu blocks and their dependency on nesting.

Consider the PDD for the "IBM LaserPrinter 4039 16L":

```
menus "pcl_banner_options" {<-- line #1
      title       "PCL Banner Options"
      prompt      "Banner Options"
            help        "PCL_Banner_Options_Help"
            next_ptr"none"

            sub_list"pcl_banner_page"
                        .
                        .
                        .
            sub_string"pcl_banner_file"
                        .
                        .
                        .
            sub_number"pcl_banner_page_length"
                        .
                        .
                        .
      }

      menus "ps_banner_options" {         <-- line #2
            title       "PS Banner Options"
            prompt      "Banner Options"
            help        "Banner_Options_Menu_Help"
            next_ptr    "none"

            sub_list    "ps_banner_page"
                        .
                        .
                        .
            sub_string  "ps_banner_file"
                        .
                        .
                        .
      }
```

```
menus "pcl_options" {                    <-- line #3
      title       "PCL Options"
      prompt      "PCL Options"
      help        "PCL_Options_Menu_Help"
      next_ptr    "install_queue()"

      sub_list    "pcl_duplex"       <-- line #4
                  .
                  .
                  .
                  .
                  .
      sub_menu    "pcl_banner_options" <-- line #5
}

menus "ps_options" {                     <-- line #6
      title       "PS Options"
      prompt      "PS Options"
      help        "PS_Options_Menu_Help"
      next_ptr  "install_queue()"

      sub_list  "ps_resolution"
                  .
                  .
                  .
      sub_menu  "ps_banner_options"     <-- line #7
}

menus "auto_options" {

      title     "Automatic Data Stream Sniffing Options"
      prompt    "This is automatic data stream options Menu"
      help      "Auto_Options_Menu_Help"
      next_ptr  "install_queue()"

      sub_menu  "pcl_options"        <-- line #8
      sub_menu  "ps_options"         <-- line #9
}
```

The above example describes the MENU blocks in good detail. The keywords for this block are

```
sub_list    STRING

sub_string  STRING

sub_number  STRING

sub_menu    STRING
```

sub_<block_type> keywords are used to refer to the diffrent blocks refrenced above in the PDD file. The compiler in the first pass makes a list of all these tags and tries to match it to the list of already parsed tags and verify the tag name and the tag type. If a tag is non-existant or its type seems to not match it will print out an error message.

EXAMPLE:

```
sub_list "pcl_duplex"
```

On line #4, must have appeared as a valid LIST block in the file before this line. If *"pcl_duplex"* was not a LIST block and according to line #4 it is a LIST block because of the keyword *"sub_list"*, the compiler will produce and error message stating that

```
Checking syntax ...

Warning: block 'pcl_duplex' seems to have incorrect
         TYPE, I think it should be: STRING

1583: Missing pre-defined STRING block at
      '"pcl_duplex"'

Compilation aborted.
```

The above message displays the location of the error. The compiler assumes that the entry in the menu block is correct, therefore, if the entry in the menu block said

```
sub_string "pcl_duplex"
```

and the *"pcl_duplex"* was found to be a LIST block, the compiler as shown above will assume that the menu block is correct and suggests that

```
I think it should be: STRING
```

This will indicate an error situation, which can be investigated by the programmer and should be corrected before proceeding any further. The line number of the error, is generally a close approximation of the area of the error. Look over and below that line number to check the other surrounding lines.

EXAMPLE:

Every sub_* should be defined before it is referenced in the MENU block. If `sub_*` block is not found

```
Checking syntax ... Failed.

1907: Missing pre-defined NUMBER block at
      '"pcl_page_length'Compilation aborted.
```

This points at the missing NUMBER block which should exist before it can be refrenced by

```
sub_number  "pcl_page_length"
```

line in the MENU block.

EXAMPLE:

Nesting of menus can be seen from the above example. Refere to line #1

Miscellaneous Rules

- If a block is refrenced in the sub_menu, sub_list, sub_string or sub_number field, it has to be defined before it is being called.
- The compiler will abort if the sub_* block is not found and / or the type is not correct, i.e. STRING, LIST, etc.
- The sequence of fields in the option_type block is important. If the sequence is not correct it will result in a syntax error.

- The block referenced by the "next_ptr" field should be found at the end of the parsing. There is no restriction of where it can be placed in the file or what is the type (i.e. LIST, STRING, NUMBER)
- A comment starts at the begining of the line with '#' character. Anything after that on that line is considered as a comment and is ignored.
- Tabs, spaces, and comments are ignored.
- Only the MENU can have multiple nesting levels.
- Data is written to the disk in the same sequence as it is read. The structure is created when it is read from the binary file.
- The Printer Definition Database (PDD) is started with the keyword "pdd_file".