The widows-and-orphans package

Frank Mittelbach

Abstract

The widows-and-orphans package checks page or column breaks for issues with widow or orphan lines and issues warnings if such problems are detected. In addition, it checks and complains about breaks involving hyphenated words and warns about display formulas directly after a page break — if they are allowed by the document parameter settings, which by default isn't the case.

A general discussion of the problem of widows and orphans and suggestions for resolution is given in [1].

Contents

1 Overview

To determine if a widow or orphan has occurred at a column or page break, the package analyzes the `\outputpenalty` that triggered the break. As TEX adds special penalties to widow and orphan lines (`\widowpenalty` and `\clubpenalty`), we can hope to identify them, provided the penalties have unique values so that we don't end up with false positives.

The package therefore analyzes the different parameter values and if necessary adjusts the settings slightly so that all possible combinations that can appear in documents have unique values and can thus be identified.

All that remains is to hook into the output routine check; if `\outputpenalty` has a value that matches one of the problematic cases, issue a warning.

Besides widows and orphans it is possible to detect other cases controlled through penalty parameters, e.g., `\brokenpenalty` that is added if a line ends in a hyphen. So by including this parameter into the checks, we can identify when that happens at the end of a column and issue a warning there too.

We also do this for `\predisplaypenalty`, which controls a break just in front of a math display. This is normally set to `10000` so such breaks don't happen in standard LATEX, but if the value is lowered it becomes possible, and thus a possible issue.

1.1 Options

The package has a number of key/value options to adjust its behavior. The option `check` defines what happens when an issue is found: default is `warning`, other possibilities are `error`, `info` and `none`.

The options `orphans` and `widows` set reasonable parameter values; the default is to use whatever the class defines. Possible values are `prevent`, `avoid` or `default`, the latter meaning use standard LATEX defaults.

To set all parameters in one go you can use `prevent-all`, `avoid-all` or `default-all`. These options also assign values to `\brokenpenalty` and `\predisplaypenalty`.

1.2 User commands

The package provides three user-level commands.

\WaOsetup

`\WaOsetup ⟨comma list⟩`

This command accepts any of the package options and allows adjusting the package behavior in mid-document if necessary.

\WaOparameters

`\WaOparameters`

This command produces a listing of the parameter combinations and their values.

\WaOignorenext

`\WaOignorenext`

This command directs the package to not generate warnings for the current page or columns (if we know that they can't be corrected).

1.3 Related packages

Package nowidow: This package offers some commands to help pushing lines from one page to another by locally requesting no widows or orphans — possibly for several lines. In that respect it implements one of the possibilities discussed in the *TUGboat* article [1]. This is, however, in many cases not the best solution to get rid of a widow or orphan when the interest is to achieve high typographical quality.

2 The implementation

The package is implemented in expl3, so the first thing we do is to define a prefix for our internal commands, so that we can write `@@` in the source when we mean the prefix `__fmwao`, indicating that something is internal.[1]

```
⟨@@=fmwao⟩
```

Then we check that we are running on top of LaTeX 2_ε and load the two packages we want to use: xparse for the user interface and l3keys2e for key/value option syntax. They load the needed expl3 code so we are ready to roll afterwards.

```
\NeedsTeXFormat{LaTeX2e}    \RequirePackage{xparse,l3keys2e}
```

Then we announce the package to the world at large. This declaration will also tell LaTeX that this is an expl3 package and that from now on the expl3 conventions are to be obeyed, e.g., _ and : can appear in command names and whitespace is automatically ignored (so no need for % all over the place).[2]

```
\ProvidesExplPackage{widows-and-orphans}{2018/11/18}{v1.0b}
                        {Detecting widows and orphans (FMi)}
```

\@makecol

As mentioned in the introduction we want to check the value of `\outputpenalty` inside the output routine, and so we need to add some code to the output routine.

We add it to the front of `\@makecol`, the macro that assembles the actual column. This way it only gets executed if we make a column or a page but not when the output routine is triggered because of a float or `\marginpar`.

```
\tl_put_left:Nn \@makecol { \__fmwao_test_for_widows_etc: }
```

(*End definition for* `\@makecol`.)

[1] `l3docstrip` expands that for us, so in the `.sty` file we get the longer names with double underscores even though the real source just contains `@@` all over the place. The same is done by the `l3doc` class for the printed documentation you are currently reading.

[2] Figure 1 gives a short introduction to the naming conventions of expl3 for those readers who haven't seen any code written in expl3. For more details refer to [2].

Commands in expl3 use the following naming convention:

```
    \⟨module⟩_⟨action⟩:⟨arg-spec⟩    % externally available command
  \__⟨module⟩_⟨action⟩:⟨arg-spec⟩    % internal command local to the package
```

⟨*module*⟩ describes the (main) area to which the command belongs,

⟨*action*⟩ describes the (main) action of the command, and

⟨*arg-spec*⟩ shows the expected command arguments and their preprocessing:

- `N` means expect a single token;
- `n` means expect a (normal) braced argument;
- `T` and `F` also represent braced arguments, indicating "true" and "false" branches in a conditional;
- `V` means expect a single token, interpret it as a variable and pass its value on to the command;
- Finally, `p` stands for a (possibly empty) parameter spec, e.g., `#1#2...` in a definition.
- There are a number of other argument types, but they aren't used in the code described here.

Examples:

`\cs_if_eq:NNTF` is a conditional from the module `cs` (command names) implementing the action `if_eq` (if equal) and expecting two single tokens (commands to compare) and a true and a false branch (one of which is executed).

`\tl_put_left:Nn` is a function from the module `tl` (token lists) implementing `put_left` and expecting a single token (a token list variable) and a braced argument (the data to insert at the front/left in the variable).

Variables start with `\l_` (for local) or `\g_` (for global) and have the data type as the last part of the name. Variables internal to the package use two underscores, e.g., `\l__fmwao_gen_warn_bool`.

Figure 1: Crash course in expl3 command name conventions

2.1 Checking `\outputpenalty`

`\g__fmwao_gen_warn_bool` To be able to suppress checking we define a global boolean variable which by default is set to `true` (warnings enabled).

```
\bool_new:N \g__fmwao_gen_warn_bool
\bool_gset_true:N \g__fmwao_gen_warn_bool
```

(*End definition for* `\g__fmwao_gen_warn_bool`.)

`\__fmwao_test_for_widows_etc:` What are the different values related to orphans and widows and the like that can appear in `\outputpenalty`? Here is the basic list:

`\widowpenalty` + `\interlinepenalty` → if the break happens on the second-last line of a paragraph and the paragraph is not followed by a math display.

`\displaywidowpenalty` + `\interlinepenalty` → if the break happens on the second-last line of a paragraph and the paragraph is followed by a math display.

`\clubpenalty` + `\interlinepenalty` → if the break happens after the first line of a paragraph and the paragraph has more than two lines.

`\clubpenalty` + `\widowpenalty` + `\interlinepenalty` → if the break happens after the first line of a paragraph in a two-line paragraph (thus this line is also the second-last line).

`\clubpenalty` + `\displaywidowpenalty` + `\interlinepenalty` → if the break happens after the first line of a paragraph in a two-line paragraph and a math display follows.

That's it for widows and orphans. If we also consider hyphenated breaks then we get a further set of cases, namely all of the above with `\brokenpenalty` added in and the case of `\brokenpenalty` on its own (with just `\interlinepenalty` added).

Frank Mittelbach

```
\cs_new:Npn \__fmwao_test_for_widows_etc: {
```

So here is the main test. We compare `\outputpenalty` with each case until we have a hit or run out of cases. Instead of adding `\interlinepenalty` to each case we subtract it once from `\outputpenalty` to make the comparison a little more efficient.

If we get a hit, we execute the corresponding code: either `\__fmwao_problem_identified:n` or `\__fmwao_problem_identified:nn` (i.e., one or two arguments) to issue the warning. The arguments are to select the correct warning text and customize it further if necessary. For example, in the first case it is a "widow" problem and the text in the message should start with "`Widow`" while in the second case it is also a "widow" problem but the text will say "`Display~ widow`".[3] This just saves a bit of space when different cases share more or less the same message text.

```
  \int_case:nnF { \outputpenalty - \interlinepenalty }
    {
      { \widowpenalty }
        { \__fmwao_problem_identified:nn{widow}{Widow} }
      { \displaywidowpenalty }
        { \__fmwao_problem_identified:nn{widow}{Display~ widow} }
      { \clubpenalty }
        { \__fmwao_problem_identified:n{orphan} }
      { \clubpenalty + \widowpenalty }
        { \__fmwao_problem_identified:nn{orphan-widow}{} }
      { \clubpenalty + \displaywidowpenalty }
        { \__fmwao_problem_identified:nn{orphan-widow}{display} }
```

A similar issue comes from a hyphen at the end of a column or page in which case TEX adds `\brokenpenalty` so we can test against that:

```
      { \brokenpenalty }
        { \__fmwao_problem_identified:n{hyphen} }
```

However, I said "TEX adds", which means if a widow line also ends in a hyphen then the penalty will be the sum of both individual penalties. So all the cases above need to be repeated with `\brokenpenalty` added to the value. We generate the same warnings, though — e.g., we will say "Widow detected" and not "Hyphenated widow line detected" as the latter seems to be overkill.

```
      { \brokenpenalty + \widowpenalty }
        { \__fmwao_problem_identified:nn{widow}{Widow} }
      { \brokenpenalty + \displaywidowpenalty }
        { \__fmwao_problem_identified:nn{widow}{Display~ widow} }
      { \brokenpenalty + \clubpenalty }
        { \__fmwao_problem_identified:n{orphan} }
      { \brokenpenalty + \clubpenalty + \widowpenalty }
        { \__fmwao_problem_identified:nn{orphan-widow}{} }
      { \brokenpenalty + \clubpenalty + \displaywidowpenalty }
        { \__fmwao_problem_identified:nn{orphan-widow}{display} }
```

Finally there is `\predisplaypenalty` that we may as well check also (in case it was set to a value lower than `10000`). If it appears it means we have a display at the very top of the page. We reuse the "widow" warning but this time say "`Lonely~ display`" in the second argument. This case does not have `\interlinepenalty` added by TEX, so we have to undo the optimization above.

```
      { \predisplaypenalty - \interlinepenalty }
        { \__fmwao_problem_identified:nn{widow}{Lonely~ display} }
    }
```

[3] If you haven't seen much expl3 code you may wonder about the `~`. As the code ignores spaces we have to mark up real spaces and for this the tilde is used. In expl3 code this does not act as a tie, but simply as a catcode 10 space character (while normal spaces are ignored).

The last argument of `\int_case:nnF` is executed in the "false" case, i.e., when no match has been found. In that case we check the status of `\g__fmwao_gen_warn_bool` and if that is also "false", i.e., we have been asked not to generate warnings, we issue an error message. Why? Because the user asked us explicitly to ignore problems on the current page, but we found nothing wrong. This either means a problem got corrected or the request was intended for a different page. Either way it is probably worth checking.

```
    { \bool_if:NF \g__fmwao_gen_warn_bool
        { \msg_error:nn{widows-and-orphans}{no-problem} } }
```

Finally, we make sure that the next page or column is again checked.

```
  \bool_gset_true:N \g__fmwao_gen_warn_bool
}
```

(*End definition for* `\__fmwao_test_for_widows_etc:`.)

`\__fmwao_problem_identified:n`
`\__fmwao_problem_identified:nn`

These commands prepare for generating a warning, but only if we are supposed to, i.e., if `\g__fmwao_gen_warn_bool` is true.

```
\cs_new:Npn \__fmwao_problem_identified:n #1 {
   \bool_if:NT \g__fmwao_gen_warn_bool
               { \msg_warning:nn{widows-and-orphans}{#1} }
}
\cs_new:Npn \__fmwao_problem_identified:nn #1 #2 {
   \bool_if:NT \g__fmwao_gen_warn_bool
               { \msg_warning:nnn{widows-and-orphans}{#1}{#2} }
}
```

(*End definition for* `\__fmwao_problem_identified:n` *and* `\__fmwao_problem_identified:nn`.)

2.2 Messages to the user

`\__fmwao_this_page:`
`\__fmwao_next_page:`

For displaying nice messages to the user we need a few helper commands. The two here show the page number of the current or next page. They are semi-smart, that is they will recognize if the document uses roman numerals and if so display the number as a roman numeral (but in all other cases it uses arabic numerals).

```
\cs_new:Npn \__fmwao_this_page: { \__fmwao_some_page:n   \c@page       }
\cs_new:Npn \__fmwao_next_page: { \__fmwao_some_page:n { \c@page + 1 } }
```

(*End definition for* `\__fmwao_this_page:` *and* `\__fmwao_next_page:`.)

`\__fmwao_some_page:n`
`\__fmwao_roman_thepage:`

This macro first compares `\thepage` against the code that would be used in the case of a roman numeral representation, and then displays its argument using either arabic numbers or roman numerals.

```
\cs_new:Npn \__fmwao_some_page:n #1 {
  \cs_if_eq:NNTF \thepage \__fmwao_roman_thepage:
    { \int_to_roman:n } { \int_to_arabic:n }
  { #1 }
}
```

`\__fmwao_roman_thepage:` just stores the default definition of `\thepage` if page numbers are represented by roman numerals for use in the comparison above.

```
\cs_new_nopar:Npn \__fmwao_roman_thepage: {\csname @roman\endcsname \c@page}
```

(*End definition for* `\__fmwao_some_page:n` *and* `\__fmwao_roman_thepage:`.)

`\legacy_switch_if:nTF`

To evaluate LaTeX 2$_\varepsilon$ boolean switches in a nice way, we need a conditional. Eventually this will probably make it into the expl3 code in this or a similar form, but right now it is missing.

```
\prg_new_conditional:Npnn \legacy_switch_if:n #1 {p, T , F , TF }
  { \exp_args:Nc\if_meaning:w { if#1 } \iftrue \prg_return_true:
                                       \else: \prg_return_false: \fi: }
```

(*End definition for* `\legacy_switch_if:nTF`.)

The first message is issued if we have been directed to ignore a problem and there wasn't one:

```
\msg_new:nnnn {widows-and-orphans} {no-problem}
  { No~ problem~ to~ suppress~ on~ this~ page! }
  { Suppression~ of~ a~ widow~ or~ orphan~ problem~ was~ requested~
    but~ on~ the~ current~ page~ there~ doesn't~ seem~ to~ be~ any.~
    Maybe~ the~ text~ was~ changed~ and~ the~ request~ should~ get~
    (re)moved?}
```

The next message is about orphans. They can appear at the bottom of the first or the second column of the current page, if we are in two-column mode. So we check for this and adjust the message accordingly.

```
\msg_new:nnnn {widows-and-orphans} {orphan}
  { Orphan~ on~ page~ \__fmwao_this_page:
    \legacy_switch_if:nT {@twocolumn}
      { \space ( \legacy_switch_if:nTF {@firstcolumn}
                                      { first~ } { second~ } column) }
  }
  { Check~ out~ the~ page~ and~ see~ if~ you~ can~ avoid~ the~ orphan.}
```

A hyphen at the end of a page or column requires more or less the same message, so this could have been combined with the previous one.

```
\msg_new:nnnn {widows-and-orphans} {hyphen}
  { Hyphen~ in~ last~ line~ of~ page~ \__fmwao_this_page:
    \legacy_switch_if:nT {@twocolumn}
      { \space ( \legacy_switch_if:nTF {@firstcolumn}
                                      { first~ } { second~ } column) }
  }
  { Check~ out~ the~ page~ and~ see~ if~ you~ can~ get~
    a~ better~ line~ break. }
```

Widows need a different logic since we detect them when cutting a previous page or column but the widow is on the following one. This message works for "widows", "display widows" as well as math displays by just changing the first word (or words), so here we use an additional argument:

```
\msg_new:nnnn {widows-and-orphans} {widow}
  { #1~ on~ page~
    \legacy_switch_if:nTF {@twocolumn}
      { \legacy_switch_if:nTF {@firstcolumn}
          { \__fmwao_this_page: \space (second~ }
          { \__fmwao_next_page: \space (first~  }
        column)
      }
      { \__fmwao_next_page: }
  }
  { Check~ out~ the~ page~ and~ see~ if~ you~ can~ avoid~ the~ widow.}
```

The case of both widow and orphan is similar, but we obviously need different text so we made it its own message.

```
\msg_new:nnnn {widows-and-orphans} {orphan-widow}
  { Orphan~
    \legacy_switch_if:nTF {@twocolumn}
       { \legacy_switch_if:nTF {@firstcolumn}
           { and~ #1 widow~ on~ page~ \__fmwao_this_page: \space
             (first~ and~ second~ }
           { on~ page~ \__fmwao_this_page: \space (second~ column)~
             and~ #1 widow~ on~ page~ \__fmwao_next_page: \space (first~ }
       }
       { on~ page~ \__fmwao_this_page: \space (second~ column)~
         and~ #1 widow~ on~ page~ \__fmwao_next_page: \space (first~ }
     column)
  }
  { Check~ out~ the~ page~ and~ see~ if~ you~ can~ avoid~ both~
    orphan~ and~ widow.}
```

2.3 Adjusting parameter values

To avoid (a lot of) false positives during checking it is important that the parameter values are chosen in a way that all possible combinations lead to unique `\outputpenalty` values. At the same time, we want them to be as close as possible to the values that have been initially requested by the user (or in the document class) and if we deviate too much then this will likely alter the page breaks TeX finds. So here is an outline of how we handle the parameters:

- We set up a property list to hold penalty values that can appear in `\outputpenalty` inside the output routine. The penalties are the "keys" and the corresponding property list value is the source of how they got produced. For example, the key might be `150` and the value `\widowpenalty`.
- Initially the property list is empty. So adding the first item simply means taking the value of one parameter, say `150` from `\widowpenalty` + `\interlinepenalty`, as the key and this formula as the property list value.
- For the next parameter, say `\clubpenalty`, we check if its value (or more precisely its value plus `\interlinepenalty`) is already a key in the property list. If that is the case, then we have failed and must modify the parameter value somehow.
- If not, we also have to check any combination of the current parameter with any parameter processed earlier. If that combination is possible, e.g., `\clubpenalty` (new) and `\widowpenalty` (already processed) then we also have to check the sum. If that sum is already a key in the property list then we have failed as well.
- If we have failed, we iterate by incrementing the current parameter value and try again. Eventually we will get to a value where all combinations we test work, that is, are not yet in the property list.
- We then change the parameter to this value and add all the combinations we tried before to the property list (that is `\clubpenalty` + `\interlinepenalty` both alone and together with `\widowpenalty` in our example). Thus from now on those are also forbidden values.
- We do all this with a helper command that takes the new parameter as the first argument and the list of different cases to try as a comma-separated list as a second argument, e.g.,

  ```
  \__fmwao_decide_penalty:Nn \clubpenalty
            { \clubpenalty + \interlinepenalty ,
              \clubpenalty + \widowpenalty + \interlinepenalty }
  ```

- This way we are adding all relevant parameters to the property list and at the same time adjusting their values if needed.

- Once all parameters are handled the property list is no longer needed as the parameters got changed along the way, but we keep it around as it allows for a simple display of all settings in one go.

`\l__fmwao_penalties_prop` Here is the property list for our process.

```
\prop_new:N \l__fmwao_penalties_prop
```

(*End definition for* `\l__fmwao_penalties_prop`.)

`\__fmwao_initialize:` Now we are ready to go. The first action is to clear the property list as the initialization may happen several times.

```
\cs_new:Npn \__fmwao_initialize: {
  \prop_clear:N  \l__fmwao_penalties_prop
```

When TeX breaks a page at a glue item with no explicit penalty involved it sets `\outputpenalty` to `10000` in the output routine to distinguish it from a case where an explicit penalty of `0` was in the document. That means none of our parameters or parameter combinations can be allowed to have that particular value, because otherwise we would get a false match for each break at glue and report an issue. So we enter that value first (by hand) so that it will not be used by a parameter or parameter combination.

```
  \prop_put:Nnn \l__fmwao_penalties_prop {10000} {break~ at~ glue}
```

The next thing is to add the values for `\@lowpenalty`, `\@medpenalty`, `\@highpenalty` to the property list as they also may show up in `\outputpenalty` if a user says, for example, `\nopagebreak[2]`.

Such a penalty from an explicit page break request does not get `\interlinepenalty` added in.

```
  \__fmwao_decide_penalty:Nn \@lowpenalty  { \@lowpenalty}
  \__fmwao_decide_penalty:Nn \@medpenalty  { \@medpenalty}
  \__fmwao_decide_penalty:Nn \@highpenalty { \@highpenalty}
```

Then comes the first real parameter for the orphans:

```
  \__fmwao_decide_penalty:Nn \clubpenalty
    { \clubpenalty + \interlinepenalty }
```

followed by the one for the widows and the one for the display widows:

```
  \__fmwao_decide_penalty:Nn \widowpenalty
    { \widowpenalty + \interlinepenalty ,
      \widowpenalty + \clubpenalty  + \interlinepenalty }

  \__fmwao_decide_penalty:Nn \displaywidowpenalty
    { \displaywidowpenalty + \interlinepenalty ,
      \displaywidowpenalty + \clubpenalty + \interlinepenalty }
```

`\brokenpenalty` can appear on its own, and also with each and every combination we have seen so far:

```
  \__fmwao_decide_penalty:Nn \brokenpenalty
    { \brokenpenalty + \interlinepenalty ,
      \brokenpenalty + \clubpenalty + \interlinepenalty ,
      \brokenpenalty + \widowpenalty + \interlinepenalty ,
      \brokenpenalty + \widowpenalty + \clubpenalty + \interlinepenalty ,
      \brokenpenalty + \displaywidowpenalty + \clubpenalty
                                          + \interlinepenalty }
```

Finally we have the parameter for lonely displays (again without `\interlinepenalty` being added):

```
  \__fmwao_decide_penalty:Nn \predisplaypenalty { \predisplaypenalty }
}
```

If we run the above code with LaTeX's default parameter settings in force it will make a few adjustments and the property list will afterwards contain the following entries:

```
The property list \l__fmwao_penalties_prop contains the pairs (without
outer braces):
>  {10000}  =>  {break at glue}
>  {51}  =>  {\@lowpenalty }
>  {151}  =>  {\@medpenalty }
>  {301}  =>  {\@highpenalty }
>  {150}  =>  {\clubpenalty +\interlinepenalty }
>  {152}  =>  {\widowpenalty +\interlinepenalty }
>  {302}  =>  {\widowpenalty +\clubpenalty +\interlinepenalty }
>  {50}  =>  {\displaywidowpenalty +\interlinepenalty }
>  {200}  =>  {\displaywidowpenalty +\clubpenalty +\interlinepenalty }
>  {100}  =>  {\brokenpenalty +\interlinepenalty }
>  {250}  =>  {\brokenpenalty +\clubpenalty +\interlinepenalty }
>  {252}  =>  {\brokenpenalty +\widowpenalty +\interlinepenalty }
>  {402}  =>  {\brokenpenalty +\widowpenalty +\clubpenalty +\interlinepenalty }
>  {300}  =>  {\brokenpenalty +\displaywidowpenalty +\clubpenalty
                                                            +\interlinepenalty }
>  {10001}  =>  {\predisplaypenalty }.
```

(*End definition for* `\__fmwao_initialize:`.)

`\l__fmwao_tmp_int` `\l__fmwao_tmp_tl` `\l__fmwao_success_bool`

For doing the calculations and insertions into the property list, we will also need an integer register, a token list variable and another boolean variable.

```
\int_new:N  \l__fmwao_tmp_int
\tl_new:N   \l__fmwao_tmp_tl
\bool_new:N \l__fmwao_success_bool
```

(*End definition for* `\l__fmwao_tmp_int`*,* `\l__fmwao_tmp_tl`*, and* `\l__fmwao_success_bool`*.*)

`\__fmwao_decide_penalty:Nn`

This is the core command that does the real work of choosing values. Let's recall that its first argument is the parameter we are currently handling and the second argument is a comma-separated list of cases for which we need to ensure that their results are not yet in the property list.

```
\cs_new:Npn \__fmwao_decide_penalty:Nn #1 #2 {
```

We start by setting the boolean to `false` and then run a loop until we have found a suitable parameter value that meets our criteria.

```
  \bool_set_false:N \l__fmwao_success_bool
  \bool_do_until:Nn \l__fmwao_success_bool
```

Inside the loop we start with the assumption that the current value of the parameter is fine and then check if that assumption is true. If yes, we can exit the loop, otherwise we will have to try with a different value.

```
   { \bool_set_true:N \l__fmwao_success_bool
```

For the verification we try each item in the second parameter to see if that is already in the property list. This means evaluating the expression to get the penalty value and then looking it up in the property list. If it is there, we have failed. In this case we set the boolean back to false and break out of the loop over the second argument since there is no point in testing further.

```
     \clist_map_inline:nn { #2 }
       { \int_set:Nn \l__fmwao_tmp_int {##1}
         \prop_get:NVNT
            \l__fmwao_penalties_prop \l__fmwao_tmp_int \l__fmwao_tmp_tl
            { \clist_map_break:n {\bool_set_false:N\l__fmwao_success_bool} }
       }
```

Once we have finished, the boolean will tell us if we are successful so far. If yes, it means there was no conflict. We therefore add all combinations with this parameter to the property list, as from now on they are forbidden as well.

So we map once more over the second argument and enter them:

```
      \bool_if:NTF \l__fmwao_success_bool
        { \clist_map_inline:nn { #2 }
            { \int_set:Nn \l__fmwao_tmp_int {##1}
              \prop_put:NVn \l__fmwao_penalties_prop \l__fmwao_tmp_int {##1}
            } }
```

If we failed we increment the parameter value and retry:

```
        { \int_incr:N #1 }
      }
}
```

One place where we will run this code is at the beginning of the document (so that changes to the parameters in the document class or the preamble are picked up). The other place is when the user changes any of the parameters in the middle of the document via `\WaOsetup`.

```
\AtBeginDocument { \__fmwao_initialize: }
```

(*End definition for* `\__fmwao_decide_penalty:Nn`.)

2.4 The option setup

The options are fairly straightforward:

```
\keys_define:nn {fmwao} {
```

By default messages are given as warnings above. If anything else is wanted the option `check` can be used which simply changes the message class used internally:

```
  ,check .choice:
  ,check / error
     .code:n = \msg_redirect_module:nnn {widows-and-orphans}{warning}{error}
  ,check / info
     .code:n = \msg_redirect_module:nnn {widows-and-orphans}{warning}{info}
  ,check / none
     .code:n = \msg_redirect_module:nnn {widows-and-orphans}{warning}{none}
  ,check / warning
     .code:n = \msg_redirect_module:nnn {widows-and-orphans}{warning}{ }
```

The other options set parameters to some hopefully “reasonable” values — no real surprises here. LaTeX internally uses `\@clubpenalty` so we need to set this too, if we change `\clubpenalty`.

```
  ,orphans .choice:
  ,orphans / prevent .code:n = \int_set:Nn \clubpenalty  { 10000        }
                               \int_set:Nn \@clubpenalty { \clubpenalty }
  ,orphans / avoid   .code:n = \int_set:Nn \clubpenalty  {  5000        }
                               \int_set:Nn \@clubpenalty { \clubpenalty }
  ,orphans / default .code:n = \int_set:Nn \clubpenalty  {   150        }
                               \int_set:Nn \@clubpenalty { \clubpenalty }

  ,widows .choice:
  ,widows / prevent  .code:n = \int_set:Nn \widowpenalty  { 10000   }
  ,widows / avoid    .code:n = \int_set:Nn \widowpenalty  {  5000   }
  ,widows / default  .code:n = \int_set:Nn \widowpenalty  {   150   }

  ,hyphens .choice:
  ,hyphens / prevent .code:n = \int_set:Nn \brokenpenalty { 10000   }
  ,hyphens / avoid   .code:n = \int_set:Nn \brokenpenalty {  2000   }
  ,hyphens / default .code:n = \int_set:Nn \brokenpenalty {    50   }
  ,prevent-all  .code:n = \int_set:Nn \clubpenalty       { 10000   }
                          \int_set:Nn \widowpenalty      { 10000   }
                          \int_set:Nn \displaywidowpenalty{ 10000   }
                          \int_set:Nn \brokenpenalty     { 10000   }
                          \int_set:Nn \predisplaypenalty  { 10000   }
                          \int_set:Nn \@clubpenalty  { \clubpenalty }
```

As an exception, `avoid-all` doesn't set `\predisplaypenalty`; maybe it should.

```
  ,avoid-all    .code:n = \int_set:Nn \clubpenalty        { 5000   }
                          \int_set:Nn \widowpenalty       { 5000   }
                          \int_set:Nn \displaywidowpenalty { 2000   }
                          \int_set:Nn \brokenpenalty      { 2000   }
%                         \int_set:Nn \predisplaypenalty   { 9999   }
                          \int_set:Nn \@clubpenalty  { \clubpenalty }
```

`default-all` reverts back to the standard LaTeX default values:

```
  ,default-all  .code:n = \int_set:Nn \clubpenalty         { 150   }
                          \int_set:Nn \widowpenalty        { 150   }
                          \int_set:Nn \displaywidowpenalty  {  50   }
                          \int_set:Nn \brokenpenalty       { 100   }
                          \int_set:Nn \predisplaypenalty  { 10000   }
                          \int_set:Nn \@clubpenalty  { \clubpenalty }
}
```

Once declared we evaluate the options given to the package:

```
\ProcessKeysPackageOptions{fmwao}
```

2.5 Document-level commands

Finally we declare the user-level commands:

`\WaOsetup` This runs the key setup on the first argument and then reinitializes the parameter setup:

```
\NewDocumentCommand\WaOsetup{m}
  { \keys_set:nn{fmwao}{#1}  \__fmwao_initialize: \ignorespaces }
```

(*End definition for* `\WaOsetup`. *This function is documented on page 253.*)

`\WaOparameters` This parameterless command outputs a display of the current parameter settings.

```
\NewDocumentCommand\WaOparameters{}{\prop_show:N \l__fmwao_penalties_prop}
```

(*End definition for* `\WaOparameters`. *This function is documented on page 253.*)

`\WaOignorenext` And here is the command that suppresses any warning on the current page or column:

```
\NewDocumentCommand\WaOignorenext{}
  { \bool_gset_false:N \g__fmwao_gen_warn_bool }
```

(*End definition for* `\WaOignorenext`. *This function is documented on page 253.*)

References

[1] Frank Mittelbach. Managing forlorn paragraph lines (a.k.a. widows and orphans) in LaTeX. *TUGboat* 39:3, 246–251, 2018.

[2] LaTeX3 Project Team. A collection of articles on expl3.
`https://latex-project.org/publications/indexbytopic/l3-expl3/`

◇ Frank Mittelbach
Mainz, Germany
`https://www.latex-project.org`
`https://ctan.org/pkg/widows-and-orphans`