This is ducksay!

(But which Version?)

(by Jonathan P. Spratte)

(Today is 2019-01-13)
## Contents

1 Documentation
   1.1 Downward Compatibility Issues .............................................. 2
   1.2 Shared between versions ..................................................... 2
      1.2.1 Macros ........................................................................... 2
      1.2.2 Options ........................................................................... 3
         1.2.2.1 Options for \AddAnimal .............................................. 4
   1.3 Version 1 ............................................................................. 5
      1.3.1 Introduction ................................................................... 5
      1.3.2 Macros ........................................................................... 5
      1.3.3 Options ........................................................................... 5
      1.3.4 Defects ........................................................................... 6
   1.4 Version 2 ............................................................................. 7
      1.4.1 Introduction ................................................................... 7
      1.4.2 Macros ........................................................................... 7
      1.4.3 Options ........................................................................... 7
   1.5 Dependencies ......................................................................... 12
   1.6 Available Animals ................................................................. 12
   1.7 Miscellaneous ....................................................................... 14

2 Implementation ......................................................................... 15
   2.1 Shared between versions ..................................................... 15
      2.1.1 Variables ....................................................................... 15
         2.1.1.1 Integers .................................................................... 15
         2.1.1.2 Sequences .................................................................. 15
         2.1.1.3 Token lists .................................................................. 15
         2.1.1.4 Boolean ..................................................................... 15
         2.1.1.5 Boxes ........................................................................ 15
      2.1.2 Regular Expressions ....................................................... 15
      2.1.3 Messages ........................................................................ 15
      2.1.4 Key-value setup ............................................................... 15
         2.1.4.1 Keys for \AddAnimal ............................................... 16
      2.1.5 Functions ........................................................................ 17
         2.1.5.1 Generating Variants of External Functions .................. 17
         2.1.5.2 Internal ...................................................................... 17
         2.1.5.3 Document level .......................................................... 18
      2.1.6 Load the Correct Version and the Animals ....................... 20
   2.2 Version 1 ............................................................................. 21
      2.2.1 Functions ....................................................................... 21
         2.2.1.1 Internal ...................................................................... 21
         2.2.1.2 Document level .......................................................... 23
   2.3 Version 2 ............................................................................. 24
      2.3.1 Messages ....................................................................... 24
      2.3.2 Variables ....................................................................... 24
         2.3.2.1 Token Lists ................................................................. 24
         2.3.2.2 Boxes ....................................................................... 24
         2.3.2.3 Bools ........................................................................ 24
         2.3.2.4 Coffins ...................................................................... 24
         2.3.2.5 Dimensions ................................................................. 24
1 Documentation

1.1 Downward Compatibility Issues

v2.0 • Versions prior to v2.0 did use a regular expression for the option ligatures, see subsubsection 1.2.2 for more on this issue. With v2.0 I do refer to the package’s version, not the code variant which can be selected with the version option.

• In a document created with package versions prior to v2.0 you’ll have to specify the option version=1 in newer versions to make those old documents behave like they used to.

v2.3 • Since v2.3 \AddAnimal and \AddColoredAnimal behave differently. You no longer have to make sure that in the first three lines every backslash which is only preceded by spaces is the bubble’s tail. Instead you can specify which symbol should be the tail and how many of such symbols there are. See subsubsection 1.2.1 for more about the current behaviour.

• The add-think key is deprecated and will throw an error starting with v2.3. In future versions it will be removed.

1.2 Shared between versions

1.2.1 Macros

A careful reader might notice that in the below list of macros there is no \ducksay and no \duckthink contained. This is due to differences between the two usable code variants (see the version key in subsubsection 1.2.2 for the code variants, subsubsection 1.3.2 and subsubsection 1.4.2 for descriptions of the two macros).

\DefaultAnimal \DefaultAnimal{(animal)}
use the (animal) if none is given in the optional argument to \ducksay or \duckthink. Package default is duck.

\DucksayOptions \DucksayOptions{(options)}
set the defaults to the keys described in subsubsection 1.2.2, subsubsection 1.3.3 and subsubsection 1.4.3. Don’t use an (animal) here, it has no effect.
\AddAnimal \AddAnimal(*\{\text{options}\}\{\text{animal}\}\{\text{ascii-art}\}
adds \text{animal} to the known animals. \text{ascii-art} is multi-line verbatim and therefore should be delimited either by matching braces or by anything that works for \verb. If the star is given \text{animal} is the new default. One space is added to the begin of \text{animal} (compensating the opening symbol). The symbols signalizing the speech bubble's tail (in the \text{hedgehog} example below the two s) can be set using the tail-symbol option and only the first tail-count occurrences will be substituted (see paragraph 1.2.2.1 for more about these options). For example, \text{hedgehog} is added with:
\AddAnimal[tail-symbol=s]{hedgehog}
{ s .\|//||
  . s /\|/||/||
  o__,_|//|/||
}
It is not checked whether the animal already exists, you could therefore redefine existing animals with this macro.

\AddColoredAnimal \AddColoredAnimal(*\{\text{options}\}\{\text{animal}\}\{\text{ascii-art}\}
It does the same as \AddAnimal but allows three different colouring syntaxes. You can use \textcolor in the \text{ascii-art} with the syntax \textcolor{\text{color}}{\text{text}}. Note that you can’t use braces in the arguments of \textcolor.
You can also use a delimited \textcolor of the form \textcolor{\text{color}}{\text{text}}\egroup, a space after that \textcolor will be considered a space in the output, you don’t have to leave a space after the \textcolor (so \textcolor{\text{color}}{\text{redText}}\textcolor{\text{color}}{\text{otherText}} is valid syntax). You can’t nest delimited \textcolors.
Also you can use an undelimited \textcolor. It affects anything until the end of the current line (or, if used inside of the \text{text} of a delimited \textcolor, anything until the end of that delimited \textcolor’s \text{text}). The syntax would be \textcolor{\text{color}}{\text{text}}.
The package doesn’t load anything providing those colouring commands for you and it doesn’t provide any coloured animals. The parsing is done using regular expressions provided by \texttt{ETP}X3. It is therefore slower than the normal \AddAnimal.

\AnimalOptions \AnimalOptions(*\{\text{animal}\}\{\text{options}\}
With this macro you can set \text{animal} specific \text{options}. If the star is given any currently set options for this \text{animal} are dropped and only the ones specified in \text{options} will be applied, else \text{options} will be added to the set options for this \text{animal}. The set \text{options} can set the tail-1 and tail-2 options and therefore overwrite the effects of \texttt{duckthink}, as \texttt{duckthink} really is just \texttt{ducksay} with the think option.

1.2.2 Options
The following options are available independent on the used code variant (the value of the version key). They might be used as package options – unless otherwise specified – or used in the macros \DucksayOptions, \ducksay and \duckthink – again unless otherwise specified. Some options might be accessible in both code variants but do slightly different things. If that’s the case they will be explained in subsubsection 1.3.3 and subsubsection 1.4.3 for version 1 and 2, respectively.

version=(number)
With this you can choose the code variant to be used. Currently 1 and 2 are available.
This can be set only during package load time. For a dedicated description of each version look into subsection 1.3 and subsection 1.4. The package author would choose version=2, the other version is mostly for legacy reasons. The default is 2.

\(\text{animal}\)\ One of the animals listed in subsection 1.6 or any of the ones added with \texttt{\textbackslash AddAnimal}. Not useable as package option. Also don’t use it in \texttt{\textbackslash DucksayOptions}, it’ll break the default animal selection.

\texttt{\textbackslash animal=\{animal\}}\ Locally sets the default animal. Note that \texttt{\textbackslash ducksay} and \texttt{\textbackslash duckthink} do digest their options inside of a group, so it just results in a longer alternative to the use of \(\text{animal}\) if used in their options.

\texttt{\textbackslash ligatures=\{token list\}}\ each token you don’t want to form ligatures during \texttt{\textbackslash AddAnimal} should be contained in this list. All of them get enclosed by grouping \{ and \} so that they can’t form ligatures. Giving no argument (or an empty one) might enhance compilation speed by disabling this replacement. The formation of ligatures was only observed in combination with \texttt{\textbackslash usepackage[T1]{fontenc}} by the author of this package. Therefore giving the option ligatures without an argument might enhance the compilation speed for you without any drawbacks. Initially this is set to ‘<>,’.\texttt{}\texttt{}

Note: In earlier releases this option’s expected argument was a regular expression. This means that this option is not fully downward compatible with older versions. The speed gain however seems worth it (and I hope the affected documents are few).

\texttt{\textbackslash no-tail}\ Sets \texttt{\textbackslash tail-1} and \texttt{\textbackslash tail-2} to be a space.

\texttt{\textbackslash say}\ Sets \texttt{\textbackslash tail-1} and \texttt{\textbackslash tail-2} as backslashes.

\texttt{\textbackslash tail-1=\{token list\}}\ Sets the first tail symbol in the output to be \(\{token list\}\). If set outside of \texttt{\textbackslash ducksay} and \texttt{\textbackslash duckthink} it will be overwritten inside of \texttt{\textbackslash duckthink} to be 0.

\texttt{\textbackslash tail-2=\{token list\}}\ Sets every other tail symbol except the first one in the output to be \(\{token list\}\). If set outside of \texttt{\textbackslash ducksay} and \texttt{\textbackslash duckthink} it will be overwritten inside of \texttt{\textbackslash duckthink} to be 0.

\texttt{\textbackslash think}\ Sets \texttt{\textbackslash tail-1}=0 and \texttt{\textbackslash tail-2}=0.

1.2.2.1 Options for \texttt{\textbackslash AddAnimal}

The options described here are only available in \texttt{\textbackslash AddAnimal} and \texttt{\textbackslash AddColoredAnimal}.

\texttt{\textbackslash tail-count=\{int\}}\ sets the number of tail symbols to be replaced in \texttt{\textbackslash AddAnimal} and \texttt{\textbackslash AddColoredAnimal}. Initial value is 2. If the value is negative every occurrence of \texttt{\textbackslash tail-symbol} will be replaced.

\texttt{\textbackslash tail-symbol=\{str\}}\ the symbol used in \texttt{\textbackslash AddAnimal} and \texttt{\textbackslash AddColoredAnimal} to mark the bubble’s tail. The argument gets \texttt{\textbackslash detokenized}. Initially a single backslash.

\[\text{(\textbackslash ~4\textbackslash ---\textbackslash \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)\]
1.3 Version 1

1.3.1 Introduction

This version is included for legacy support (old documents should behave the same without any change to them – except the usage of \texttt{version=1} as an option, for a more or less complete list of downward compatibility related problems see subsection 1.1). For the bleeding edge version of \texttt{ducksay} skip this subsection and read subsection 1.4.

1.3.2 Macros

The following is the description of macros which differ in behaviour from those of version 2.

\texttt{\ducksay\[\langle\text{options}\rangle\]}\{\langle\text{message}\rangle\}

options might include any of the options described in subsubsection 1.2.2 and subsubsection 1.3.3 if not otherwise specified. Prints an \texttt{\langle\text{animal}\rangle} saying \texttt{\langle\text{message}\rangle}. \texttt{\langle\text{message}\rangle} is not read in verbatim. Multi-line \texttt{\langle\text{message}\rangle}s are possible using \\
\texttt{\langle\text{message}\rangle}s should not be contained in a macro definition but at toplevel. Else use the option \texttt{ht}.

\texttt{\duckthink\[\langle\text{options}\rangle\]}\{\langle\text{message}\rangle\}

options might include any of the options described in subsubsection 1.2.2 and subsubsection 1.3.3 if not otherwise specified. Prints an \texttt{\langle\text{animal}\rangle} thinking \texttt{\langle\text{message}\rangle}. \texttt{\langle\text{message}\rangle} is not read in verbatim. Multi-line \texttt{\langle\text{message}\rangle}s are possible using \\
\texttt{\langle\text{message}\rangle}s should not be contained in a macro definition but at toplevel. Else use the option \texttt{ht}.

1.3.3 Options

The following options are available to \texttt{\ducksay}, \texttt{\duckthink}, and \texttt{\DucksayOptions} and if not otherwise specified also as package options:

\texttt{bubble=}\langle\text{code}\rangle

use \langle\text{code}\rangle in a group right before the bubble (for font switches). Might be used as a package option but not all control sequences work out of the box there.

\texttt{body=}\langle\text{code}\rangle

use \langle\text{code}\rangle in a group right before the body (meaning the \texttt{\langle\text{animal}\rangle}). Might be used as a package option but not all control sequences work out of the box there. E.g. to right-align the \texttt{\langle\text{animal}\rangle} to the bubble, use \texttt{body=\hfill}.

\texttt{align=}\langle\text{valign}\rangle

use \langle\text{valign}\rangle as the vertical alignment specifier given to the \texttt{tabular} which is around the contents of \texttt{\ducksay} and \texttt{\duckthink}.

\texttt{msg-align=}\langle\text{halign}\rangle

use \langle\text{halign}\rangle for alignment of the rows of multi-line \texttt{\langle\text{message}\rangle}s. It should match a \texttt{tabular} column specifier. Default is l. It only affects the contents of the speech bubble not the bubble.

\texttt{rel-align=}\langle\text{column}\rangle

use \langle\text{column}\rangle for alignment of the bubble and the body. It should match a \texttt{tabular} column specifier. Default is l.
\(wd=\langle \text{count} \rangle\) in order to detect the width the \langle \text{message} \rangle is expanded. This might not work out for some commands (e.g. \url from \texttt{hyperref}). If you specify the width using \texttt{wd} the \langle \text{message} \rangle is not expanded and therefore the command \textit{might} work out. \langle \text{count} \rangle should be the character count.

\(ht=\langle \text{count} \rangle\) you might explicitly set the height (the row count) of the \langle \text{message} \rangle. This only has an effect if you also specify \texttt{wd}.

1.3.4 Defects

- no automatic line wrapping

Ohh, no!

\[\ldots\]

\[(\ldots)\]

\[\ldots\]

\[(\ldots)\]

\[\ldots\]

\[\ldots\]

\[\ldots\]
1.4 Version 2

1.4.1 Introduction

Version 2 is the current version of ducksay. It features automatic line wrapping (if you specify a fixed width) and in general more options (with some nasty argument parsing).

If you’re already used to version 1 you should note one important thing: You should only specify the version and the ligatures during package load time as arguments to \usepackage. The other keys might not work or do unintended things and only don’t throw errors or warnings because of the legacy support of version 1. After the package is loaded, keys only used for version 1 will throw an error.

1.4.2 Macros

The following is the description of macros which differ in behaviour from those of version 1.

\ducksay[⟨options⟩]⟨message⟩

options might include any of the options described in subsubsection 1.2.2 and subsubsection 1.4.3 if not otherwise specified. Prints an ⟨animal⟩ saying ⟨message⟩. The ⟨message⟩ can be read in in four different ways. For an explanation of the ⟨message⟩ reading see the description of the arg key in subsubsection 1.4.3.

The height and width of the message is determined by measuring its dimensions and the bubble will be set accordingly. The box surrounding the message will be placed both horizontally and vertically centred inside of the bubble. The output utilizes \LaTeX3’s coffin mechanism described in interface3.pdf and the documentation of xcoffins.

\duckthink[⟨options⟩]⟨message⟩

The only difference to \ducksay is that in \duckthink the ⟨animal⟩s think the ⟨message⟩ and don’t say it.

1.4.3 Options

In version 2 the following options are available. Keep in mind that you shouldn’t use them during package load time but in the arguments of \ducksay, \duckthink or \DucksayOptions.

arg=⟨choice⟩

specifies how the ⟨message⟩ argument of \ducksay and \duckthink should be read in. Available options are box, tab and tab*:

- box: the argument is read in either as a \hbox or a \vbox (the latter if a fixed width is specified with either wd or wd*). Note that in this mode any arguments relying on category code changes like e.g. \verb will work (provided that you don’t use \ducksay or \duckthink inside of an argument of another macro of course).
- tab: the argument is read in as the contents of a tabular. Note that in this mode any arguments relying on category code changes like e.g. \verb will not work. This mode comes closest to the behaviour of version 1 of ducksay.
tab*

the argument is read in as the contents of a tabular. However it is read in verbatim and uses \scantokens to rescan the argument. Note that in this mode any arguments relying on category code changes like e.g. \verb will work. You can’t use \ducksay or \duckthink as an argument to another macro in this mode however.

b shortcut for out-v=b.

body=(font) add (font) to the font definitions in use to typeset the ⟨animal⟩’s body.

body*=⟨font⟩
clear any definitions previously made (including the package default) and set the font definitions in use to typeset the ⟨animal⟩’s body to ⟨font⟩. The package default is \verbatim@font. In addition \frenchspacing will always be used prior to the defined ⟨font⟩.

body-align=(choice)
sets the relative alignment of the ⟨animal⟩ to the ⟨message⟩. Possible choices are l, c and r. For l the ⟨animal⟩ is flushed to the left of the ⟨message⟩, for c it is centred and for r it is flushed right. More fine grained control over the alignment can be obtained with the keys msg-to-body, body-to-msg, body-x and body-y. Package default is l.

body-mirrored=(bool)
if set true the ⟨animal⟩ will be mirrored along its vertical centre axis. Package default is false. If you set it true you’ll most likely need to manually adjust the alignment of the body with one or more of the keys body-align, body-to-msg, msg-to-body, body-x and body-y.

body-to-msg=(pole)
defines the horizontal coffin ⟨pole⟩ to be used for the placement of the ⟨animal⟩ beneath the ⟨message⟩. See interface3.pdf and the documentation of xcoffins for information about coffin poles.

body-x=(dimen)
defines a horizontal offset of ⟨dimen⟩ length of the ⟨animal⟩ from its placement beneath the ⟨message⟩.

body-y=(dimen)
defines a vertical offset of ⟨dimen⟩ length of the ⟨animal⟩ from its placement beneath the ⟨message⟩.

bubble=(font)
add (font) to the font definitions in use to typeset the bubble. This does not affect the ⟨message⟩ only the bubble put around it.

bubble*=(font)
clear any definitions previously made (including the package default) and set the font definitions in use to typeset the bubble to ⟨font⟩. This does not affect the ⟨message⟩ only the bubble put around it. The package default is \verbatim@font.

bubble-bot-kern=(dimen)
specifies a vertical offset of the placement of the lower border of the bubble from the bottom of the left and right borders.

\begin{verbatim}
  (  \_\_\_
      \-\-\--
        \-\-\-
\end{verbatim}
bubble-delim-left-1=⟨token list⟩
the left delimiter used if only one line of delimiters is needed. Package default is (.

bubble-delim-left-2=⟨token list⟩
the upper most left delimiter used if more than one line of delimiters is needed. Package
default is /.

bubble-delim-left-3=⟨token list⟩
the left delimiters used to fill the gap if more than two lines of delimiters are needed.
Package default is |.

bubble-delim-left-4=⟨token list⟩
the lower most left delimiter used if more than one line of delimiters is needed. Package
default is \.

bubble-delim-right-1=⟨token list⟩
the right delimiter used if only one line of delimiters is needed. Package default is ).

bubble-delim-right-2=⟨token list⟩
the upper most right delimiter used if more than one line of delimiters is needed. Package
default is \.

bubble-delim-right-3=⟨token list⟩
the right delimiters used to fill the gap if more than two lines of delimiters are needed.
Package default is |.

bubble-delim-right-4=⟨token list⟩
the lower most right delimiter used if more than one line of delimiters is needed. Package
default is /.

bubble-delim-top=⟨token list⟩
the delimiter used to create the top and bottom border of the bubble. The package
default is {-} (the braces are important to suppress ligatures here).

bubble-side-kern=⟨dimen⟩
specifies the kerning used to move the sideways delimiters added to fill the gap for more
than two lines of bubble height. (the left one is moved to the left, the right one to the
right)

bubble-top-kern=⟨dimen⟩
specifies a vertical offset of the placement of the upper border of the bubble from the top
of the left and right borders.

c shortcut for out-v=vc.

col=⟨column⟩
specifies the used column specifier used for the ⟨message⟩ enclosing tabular for arg=tab
and arg=tab*. Has precedence over msg-align. You can also use more than one col-
umn this way: \ducksay[arg=tab,col=cc]{ You & can \ \ do & it } would be valid
syntax.

hpad=⟨count⟩
Add ⟨count⟩ times more bubble-delim-top instances than necessary to the upper and
lower border of the bubble. Package default is 2.

```
( ـ ـ ـ
\ ـ ـ
空前 ــــ ـ ــــ ـ ــــ
```
Documentation of Version 2

ht=⟨count⟩ specifies a minimum height (in lines) of the ⟨message⟩. The lines’ count is that of the needed lines of the horizontal bubble delimiters. If the count of the actually needed lines is smaller than the specified ⟨count⟩, ⟨count⟩ lines will be used. Else the required lines will be used.

ignore-body=⟨bool⟩
If set true the ⟨animal⟩’s body will be added to the output but it will not contribute to the bounding box (so will not take up any space).

msg=⟨font⟩ add ⟨font⟩ to the font definitions in use to typeset the ⟨message⟩.

msg*=⟨font⟩ clear any definitions previously made (including the package default) and set the font definitions in use to typeset the ⟨message⟩ to ⟨font⟩. The package default is \verbatim@font.

MSG=⟨font⟩ same as msg=⟨font⟩, bubble=⟨font⟩.

MSG*=⟨font⟩ same as msg*=⟨font⟩, bubble*=⟨font⟩.

msg-align=⟨choice⟩
specifies the alignment of the ⟨message⟩. Possible values are l for flushed left, c for centred, r for flushed right and j for justified. If arg=tab or arg=tab* the j choice is only available for fixed width contents. Package default is l.

msg-align-c=⟨token list⟩
set the ⟨token list⟩ which is responsible to typeset the message centred if the option msg-align=c is used. It is used independent of the arg key. For arg=tab and arg=tab* it is only used if a fixed width is specified and the macro \arraybackslash provided by array is used afterwards. The package default is \centering. It might be useful if you want to use ragged2e’s \Centering for example.

msg-align-j=⟨token list⟩
set the ⟨token list⟩ which is responsible to typeset the message justified if the option msg-align=j is used. It is used independent of the arg key. For arg=tab and arg=tab* it is only used if a fixed width is specified and the macro \arraybackslash provided by array is used afterwards. The package default is empty as justification is the default behaviour of contents of a p column and of a \vbox. It might be useful if you want to use ragged2e’s \justifying for example.

msg-align-l=⟨token list⟩
set the ⟨token list⟩ which is responsible to typeset the message flushed left if the option msg-align=l is used. It is used independent of the arg key. For arg=tab and arg=tab* it is only used if a fixed width is specified and the macro \arraybackslash provided by array is used afterwards. The package default is \raggedright. It might be useful if you want to use ragged2e’s \RaggedRight for example.

msg-align-r=⟨token list⟩
set the ⟨token list⟩ which is responsible to typeset the message flushed right if the option msg-align=r is used. It is used independent of the arg key. For arg=tab and arg=tab* it is only used if a fixed width is specified and the macro \arraybackslash provided by array is used afterwards. The package default is \raggedleft. It might be useful if you want to use ragged2e’s \RaggedLeft for example.
Documentation of Version 2

msg-to-body={pole}
defines the horizontal coffin ⟨pole⟩ to be used as the reference point for the placement of the ⟨animal⟩ beneath the ⟨message⟩. See interface3.pdf and the documentation of xcoffins for information about coffin poles.

no-bubble={bool}
If true the ⟨message⟩ will not be surrounded by a bubble. Package default is of course false.

none={bool} One could say this is a special animal. If true no animal body will be used (resulting in just the speech bubble). Package default is of course false.

out-h={pole}
defines the horizontal coffin ⟨pole⟩ to be used as the anchor point for the print out of the complete result of \ducksay and \duckthink. See interface3.pdf and the documentation of xcoffins for information about coffin poles.

out-v={pole}
defines the vertical coffin ⟨pole⟩ to be used as the anchor point for the print out of the complete result of \ducksay and \duckthink. See interface3.pdf and the documentation of xcoffins for information about coffin poles.

out-x={dimen}
specifies an additional horizontal offset of the print out of the complete result of \ducksay and \duckthink.

out-y={dimen}
specifies an additional vertical offset of the print out of the complete result of \ducksay and \duckthink

strip-spaces={bool}
if set true leading and trailing spaces are stripped from the ⟨message⟩ if arg=box is used. Initially this is set to false.

t shortcut for out-v=t.

vpad={count}
add ⟨count⟩ to the lines used for the bubble, resulting in ⟨count⟩ more lines than necessary to enclose the ⟨message⟩ inside of the bubble.

wd={count}
specifies the width of the ⟨message⟩ to be fixed to ⟨count⟩ times the width of an upper case M in the ⟨message⟩’s font declaration. A value smaller than 0 is considered deactivated, else the width is considered as fixed. For a fixed width the argument of \ducksay and \duckthink is read in as a \vbox for arg=box and the column definition uses a p-type column for arg=tab and arg=tab*. If both wd is not smaller than 0 and wd* is not smaller than 0pt, wd* will take precedence.

wd*={dimen}
specifies the width of the ⟨message⟩ to be fixed to ⟨dimen⟩. A value smaller than 0pt is considered deactivated, else the width is considered as fixed. For a fixed width the argument of \ducksay and \duckthink is read in as a \vbox for arg=box and the column definition uses a p-type column for arg=tab and arg=tab*. If both wd is not smaller than 0 and wd* is not smaller than 0pt, wd* will take precedence.
1.5 Dependencies
The package depends on the two packages `xparse` and `l3keys2e` and all of their dependencies. Version 2 additionally depends on `array` and `grabbox`.

1.6 Available Animals
The following animals are provided by this package. I did not create them (but altered some), they belong to their original creators.

*Latin; “I'm new, too.”
1.7 Miscellaneous

This work may be distributed and/or modified under the conditions of the \LaTeX{} Project Public License (LPPL), either version 1.3c of this license or (at your option) any later version. The latest version of this license is in the file: http://www.latex-project.org/lppl.txt

The package is hosted on https://github.com/Skillmon/ltx_ducksay, you might report bugs there.
2 Implementation

2.1 Shared between versions

2.1.1 Variables

2.1.1.1 Integers

\int_new:N \l_ducksay_msg_width_int
\int_new:N \l_ducksay_msg_height_int
\int_new:N \l_ducksay_tail_symbol_count_int

2.1.1.2 Sequences

\seq_new:N \l_ducksay_msg_lines_seq

2.1.1.3 Token lists

\tl_new:N \l_ducksay_align_tl
\tl_new:N \l_ducksay_msg_align_tl
\tl_new:N \l_ducksay_animal_tl
\tl_new:N \l_ducksay_body_tl
\tl_new:N \l_ducksay_bubble_tl
\tl_new:N \l_ducksay_tmpa_tl
\tl_new:N \l_ducksay_tail_symbol_out_one_tl
\tl_new:N \l_ducksay_tail_symbol_out_two_tl
\tl_new:N \l_ducksay_tail_symbol_in_tl

2.1.1.4 Boolean

\bool_new:N \l_ducksay_version_one_bool
\bool_new:N \l_ducksay_version_two_bool

2.1.1.5 Boxes

\box_new:N \l_ducksay_tmpa_box

2.1.2 Regular Expressions

Regular expressions for \AddColoredAnimal

\regex_const:Nn \c_ducksay_textcolor_regex
\regex_const:Nn \c_ducksay_color_delim_regex
\regex_const:Nn \c_ducksay_color_regex

2.1.3 Messages

\msg_new:nnn { ducksay } { load-time-only }
\msg_new:nnn { ducksay } { deprecated-key }

2.1.4 Key-value setup

\keys_define:nn { ducksay }
{ 
, bubble .tl_set:N = \l_ducksay_bubble_tl
, body .tl_set:N = \l_ducksay_body_tl
}
2.1.4.1 Keys for \AddAnimal

Define keys meant for \AddAnimal and \AddColoredAnimal only in their own regime:

\setkeys{ducksay}{version=2,add-think,think,ligatures,align=left,rel-align=left,ligatures=left,\DucksayOptions{\addleftsymbol,\addrightsymbol}}

Undefine the load-time-only keys
\setkeys{ducksay}{version=1,\DucksayOptions{\addleftsymbol,\addrightsymbol}}
\keys_define:nn { ducksay / add-animal }
{
  ,tail-symbol .code:n =
    \tl_set:Nx \l_ducksay_tail_symbol_in_tl { \tl_to_str:n { #1 } }
  ,tail-symbol .initial:o = \c_backslash_str
  ,tail-count .int_set:N = \l_ducksay_tail_symbol_count_int
  ,tail-count .initial:n = 2
}

2.1.5 Functions

2.1.5.1 Generating Variants of External Functions
\cs_generate_variant:Nn \tl_replace_once:Nnn { NVn }
\cs_generate_variant:Nn \tl_replace_all:Nnn { NVn }

2.1.5.2 Internal
\ducksay_replace_verb_newline:Nn
\cs_new_protected:Npx \ducksay_replace_verb_newline:Nn #1 #2
{
  \tl_replace_all:Nnn #1 { \char_generate:nn { 13 } { 12 } } { #2 }
}
(End definition for \ducksay_replace_verb_newline:Nn. This function is documented on page ??.)

\ducksay_replace_verb_newline_newline:Nn
\cs_new_protected:Npx \ducksay_replace_verb_newline_newline:Nn #1 #2
{
  \tl_replace_all:Nnn #1 { \char_generate:nn { 13 } { 12 } \char_generate:nn { 13 } { 12 } } { #2 }
}
(End definition for \ducksay_replace_verb_newline_newline:Nn. This function is documented on page ??.)

\ducksay_process_verb_newline:nnn
\cs_new_protected:Npn \ducksay_process_verb_newline:nnn #1 #2 #3
{
  \tl_set:Nn \ProcessedArgument { #3 }
  \ducksay_replace_verb_newline_newline:Nn \ProcessedArgument { #2 }
  \ducksay_replace_verb_newline:Nn \ProcessedArgument { #1 }
}
(End definition for \ducksay_process_verb_newline:nnn. This function is documented on page ??.)

\ducksay_add_animal_inner:nnnn
\cs_new_protected:Npn \ducksay_add_animal_inner:nnnn #1 #2 #3 #4
{
  \group_begin:
    \keys_set:nn { ducksay / add-animal } { #1 }
    \tl_set:Nn \l_ducksay_tmpa_tl \{ \ #3 \ }
    \int_compare:nNnTF { \l_ducksay_tail_symbol_count_int } < \{ \c_zero_int \ }
    { \tl_replace_once:Nnn \l_ducksay_tmpa_tl
      ( 17 )
      \ \\
      \ \ \\
      \ \\
      \ \\
      \ \\
      \ \\
      \ \\
    }
  \}
}
\begin{verbatim}
\l_ducksay_tail_symbol_in_tl
\l_ducksay_tail_symbol_out_one_tl
\tl_replace_all:Nn
\l_ducksay_tmpa_tl
\l_ducksay_tail_symbol_in_tl
\l_ducksay_tail_symbol_out_two_tl
\}
\}
\int_compare:nNnT \l_ducksay_tail_symbol_count_int > \c_zero_int
\{ \tl_replace_once:NVn
\l_ducksay_tmpa_tl
\l_ducksay_tail_symbol_in_tl
\l_ducksay_tail_symbol_out_one_tl
\int_step_inline:nnn { 2 } \l_ducksay_tail_symbol_count_int
\}
\tl_replace_once:NVn
\l_ducksay_tmpa_tl
\l_ducksay_tail_symbol_in_tl
\l_ducksay_tail_symbol_out_two_tl
\}
\tl_map_inline:Nn \l_ducksay_ligatures_tl
{ \tl_replace_all:Nnn \l_ducksay_tmpa_tl { ##1 } { { ##1 } } }
\ducksay_replace_verb_newline:Nn \l_ducksay_tmpa_tl
{ \tabularnewline
null }
\exp_args:NNnV
\group_end:
\tl_set:cn { l_ducksay_animal_#2_tl } \l_ducksay_tmpa_tl
\exp_args:Nnx \keys_define:nn { ducksay }
{ #2 .code:n =
{ \exp_not:n { \tl_set_eq:NN \l_ducksay_animal_tl }
\exp_after:wN \exp_not:N \cs:w l_ducksay_animal_#2_tl \cs_end:
\exp_not:n { \exp_args:NV \DucksayOptions }
\exp_after:wN
\exp_not:n { \exp_args:NV \cs:w l_ducksay_animal_#2_options_tl \cs_end:
} }
\}
\tl_if_exist:cF { l_ducksay_animal_#2_options_tl }
\{ \tl_new:c { l_ducksay_animal_#2_options_tl } \}
\IfBooleanT { #4 }
\{ \keys_define:nn { ducksay } { default_animal .meta:n = { #2 } } \}
\cs_generate_variant:Nn \ducksay_add_animal_inner:nnnn { nnVn }
\end{verbatim}

(End definition for \ducksay_add_animal_inner:nnnn. This function is documented on page ??.)

\subsection*{2.1.5.3 Document level}

\DefaultAnimal
\NewDocumentCommand \DefaultAnimal { m } { \keys_define:nn { ducksay } { default_animal .meta:n = { #1 } } }

(End definition for \DefaultAnimal. This function is documented on page 2.)

\DucksayOptions
\NewDocumentCommand \DucksayOptions { m } { \keys_set:nn { ducksay } { #1 } }

(End definition for \DucksayOptions. This function is documented on page 2.)

\AddAnimal
\NewDocumentCommand \AddAnimal { s O{} m +v } { \ducksay_add_animal_inner:nnnn { #2 } { #3 } { #4 } { #1 } }

(End definition for \AddAnimal. This function is documented on page 3.)

\AddColoredAnimal
\NewDocumentCommand \AddColoredAnimal { s O{} m +v } { \tl_set:Nn \l_ducksay_tmpa_tl { #4 } \regex_replace_all:NnN \c_ducksay_color_delim_regex { \c{bgroup}\c{color}\cB\{\1\cE\}\c{egroup} } \l_ducksay_tmpa_tl \regex_replace_all:NnN \c_ducksay_color_regex { \c{color}\cB\{\1\cE\} } \l_ducksay_tmpa_tl \regex_replace_all:NnN \c_ducksay_textcolor_regex { \c{textcolor}\cB\{\1\cE\}\cB\{\2\cE\} } \l_ducksay_tmpa_tl \ducksay_add_animal_inner:nnVn { #2 } { #3 } \l_ducksay_tmpa_tl { #1 } }

(End definition for \AddColoredAnimal. This function is documented on page 3.)

\AnimalOptions
\NewDocumentCommand \AnimalOptions { s m m } { \tl_if_exist:cTF { l_ducksay_animal_#2_options_tl } { \IfBooleanTF { #1 } { \tl_set:cn } { \tl_put_right:cn } } \tl_set:cn { l_ducksay_animal_#2_options_tl } { #3, } }

(End definition for \AnimalOptions. This function is documented on page 3.)
2.1.6 Load the Correct Version and the Animals

\bool_if:NT \l_ducksay_version_one_bool
{ \file_input:n { ducksay.code.v1.tex } }
\bool_if:NT \l_ducksay_version_two_bool
{ \file_input:n { ducksay.code.v2.tex } }
\ExplSyntaxOff
\input{ducksay.animals.tex}

(/pkg)
2.2 Version 1

2.2.1 Functions

2.2.1.1 Internal

\ducksay_longest_line:n  Calculate the length of the longest line
\cs_new:Npn \ducksay_longest_line:n #1
{\int_incr:N \l_ducksay_msg_height_int
 \exp_args:NNx \tl_set:Nn \l_ducksay_tmpa_tl { #1 }
 \regex_replace_all:nnN { \s } { \c { space } } \l_ducksay_tmpa_tl
 \int_set:Nn \l_ducksay_msg_width_int
 { \int_max:nn { \l_ducksay_msg_width_int } { \tl_count:N \l_ducksay_tmpa_tl } }
}

(End definition for \ducksay_longest_line:n. This function is documented on page ??.)

\ducksay_open_bubble:  Draw the opening bracket of the bubble
\cs_new:Npn \ducksay_open_bubble:
{\begin{tabular}{@{}l@{}}
\null\int_compare:nNnTF { \l_ducksay_msg_height_int } = { 1 } { ( }
{ \int_step_inline:nnn { 3 } { \l_ducksay_msg_height_int } { \\kern-0.2em| }
\null\begin{tabular}{@{}l@{}}
\_\int_step_inline:nnn { 2 } { \l_ducksay_msg_height_int } { \null\begin{tabular}{@{}r@{}}
\null\end{tabular}\null}
\end{tabular}\null}
\end{tabular}}

(End definition for \ducksay_open_bubble:. This function is documented on page ??.)

\ducksay_close_bubble:  Draw the closing bracket of the bubble
\cs_new:Npn \ducksay_close_bubble:
{\begin{tabular}{@{}l@{}}
\null\int_step_inline:nnn { 2 } { \l_ducksay_msg_height_int } { \null\begin{tabular}{@{}r@{}}
\null\end{tabular}\null}
\end{tabular}}

(End definition for \ducksay_close_bubble:. This function is documented on page ??.)
Implementation of Version 1

\int_compare:nNnTF { \l_ducksay_msg_height_int } = { 1 }
{ ) }
\detokenize { \ }
\int_step_inline:nnn
{ 3 } { \l_ducksay_msg_height_int } { \\kern-0.2em } \\/
\null
\end{tabular}

(End definition for \ducksay_close_bubble:. This function is documented on page ??.)

\ducksay_print_msg:nn Print out the message
\cs_new:Npn \ducksay_print_msg:nn #1 #2
{ \begin{tabular}{@{} #2 @{}} #1\null
\int_step_inline:nn { \l_ducksay_msg_width_int } { _ } \null
\int_step_inline:nn { \l_ducksay_msg_width_int } { { - } } \null
\end{tabular}
}
\cs_generate_variant:Nn \ducksay_print_msg:nn { nV }

(End definition for \ducksay_print_msg:nn. This function is documented on page ??.)

\ducksay_print:nn Print out the whole thing
\cs_new:Npn \ducksay_print:nn #1 #2
{ \int_compare:nNnTF { \l_ducksay_msg_width_int } < { 0 }
{ \int_zero:N \l_ducksay_msg_height_int
\seq_set_split:Nnn \l_ducksay_msg_lines_seq { \ } { #1 }
\seq_map_function:NN \l_ducksay_msg_lines_seq \ducksay_longest_line:n
}
{ \int_compare:nNnT { \l_ducksay_msg_height_int } < { 0 }
{ \regex_count:nnN { \c { \ } } { #1 } \l_ducksay_msg_width_int
\int_incr:N \l_ducksay_msg_height_int
}
}
\group_begin:
\frenchspacing
\verbatim@font
\@nolig
\begin{tabular}{\l_ducksay_align_tl}{@{}#2@{}}
\l_ducksay_bubble_tl
\begin{tabular}{@{}l@{}}
\ducksay_open_bubble:
\ducksay_print_msg:nV { #1 } \l_ducksay_msg_align_tl
\ducksay_close_bubble:
\end{tabular}
\l_ducksay_body_tl

---
\-----
\-----
\_--\
\-\-
\-

---
\-----
\-----
\_--\
\-\-
\-

---
\-----
\-----
\_--\
\-\-
\-

---
\-----
\-----
\_--\
\-\-
\-

---
\-----
\-----
\_--\
\-\-
\-

---
\-----
\-----
\_--\
\-\-
\-

---
\-----
\-----
\_--\
\-\-
\-

---
\-----
\-----
\_--\
\-\-
\-
2.2.1.2 Document level

\ducksay \NewDocumentCommand \ducksay { O{} m } { \ducksay_and_think:nn { #1 } { #2 } } (End definition for \ducksay. This function is documented on page 7.)

\duckthink \NewDocumentCommand \duckthink { O{} m } { \ducksay_and_think:nn { think, #1 } { #2 } } (End definition for \duckthink. This function is documented on page 7.)
2.3 Version 2

Load the additional dependencies of version 2.
\RequirePackage{array,grabbox}

2.3.1 Messages
\msg_new:nnn { ducksay } { justify-unavailable }
{ Justified-content-is-not-available-for-tabular-argument-mode-without-fixed-width.-'l'-column-is-used-instead. }
\msg_new:nnn { ducksay } { unknown-message-alignment }
{ The-specified-message-alignment-\exp_not:n { #1 }-is-unknown.-'l'-is-used-as-fallback. }
\msg_new:nnn { ducksay } { v1-key-only }
{ The-'\l_keys_key_tl'-key-is-only-available-for-'version=1'. }

2.3.2 Variables

2.3.2.1 Token Lists
\tl_new:N \l_ducksay_msg_align_vbox_tl

2.3.2.2 Boxes
\box_new:N \l_ducksay_msg_box

2.3.2.3 Bools
\bool_new:N \l_ducksay_eat_arg_box_bool
\bool_new:N \l_ducksay_eat_arg_tab_verb_bool
\bool_new:N \l_ducksay_mirrored_body_bool

2.3.2.4 Coffins
\coffin_new:N \l_ducksay_body_coffin
\coffin_new:N \l_ducksay_bubble_close_coffin
\coffin_new:N \l_ducksay_bubble_open_coffin
\coffin_new:N \l_ducksay_bubble_top_coffin
\coffin_new:N \l_ducksay_msg_coffin

2.3.2.5 Dimensions
\dim_new:N \l_ducksay_hpad_dim
\dim_new:N \l_ducksay_bubble_bottom_kern_dim
\dim_new:N \l_ducksay_bubble_top_kern_dim
\dim_new:N \l_ducksay_msg_width_dim

2.3.3 Options
\keys_define:nn { ducksay }
{ ,arg .choice:
, arg / box .code:n = \bool_set_true:N \l_ducksay_eat_arg_box_bool
, arg / tab .code:n =
{ \bool_set_false:N \l_ducksay_eat_arg_box_bool
 \bool_set_false:N \l_ducksay_eat_arg_tab_verb_bool
}
\begin{verbatim}
Implementation of Version 2

,\arg / tab* .code:n =
\begin{Verbatim}
{\bool_set_false:N \l_ducksay_eat_arg_box_bool
  \bool_set_true:N \l_ducksay_eat_arg_tab_verb_bool
},\arg .initial:n = tab
,\wd* .dim_set:N = \l_ducksay_msg_width_dim
,\wd* .initial:n = -\c_max_dim
,\wd* .value_required:n = true
,\none .bool_set:N = \l_ducksay_no_body_bool
,\no-bubble .bool_set:N = \l_ducksay_no_bubble_bool
,body-mirrored .bool_set:N = \l_ducksay_mirrored_body_bool
,ignore-body .bool_set:N = \l_ducksayignored_body_bool
,body-x .dim_set:N = \l_ducksay_body_x_offset_dim
,body-x .value_required:n = true
,body-y .dim_set:N = \l_ducksay_body_y_offset_dim
,body-y .value_required:n = true
,body-to-msg .tl_set:N = \l_ducksay_body_to_msg_align_body_tl
,msg-to-body .tl_set:N = \l_ducksay_body_to_msg_align_msg_tl
,\body-align .choice:
,\body-align / l .meta:n = \{ body-to-msg = l , msg-to-body = l \}
,\body-align / c .meta:n = \{ body-to-msg = hc , msg-to-body = hc \}
,\body-align / r .meta:n = \{ body-to-msg = r , msg-to-body = r \}
,\body-align .initial:n = l
,\msg-align .choice:
,\msg-align / l .code:n = \{ \tl_set:Nn \l_ducksay_msg_align_tl { l } \}
,\msg-align / c .code:n = \{ \tl_set:Nn \l_ducksay_msg_align_tl { c } \}
,\msg-align / r .code:n = \{ \tl_set:Nn \l_ducksay_msg_align_tl { r } \}
,\msg-align / j .code:n = \{ \tl_set:Nn \l_ducksay_msg_align_tl { j } \}
,\msg-align-l .tl_set:N = \l_ducksay_msg_align_l_tl
,\msg-align-l .initial:n = \raggedright
,\msg-align-c .tl_set:N = \l_ducksay_msg_align_c_tl
,\msg-align-c .initial:n = \centering
,\msg-align-r .tl_set:N = \l_ducksay_msg_align_r_tl
,\msg-align-r .initial:n = \raggedleft
,\msg-align-j .tl_set:N = \l_ducksay_msg_align_j_tl
,\msg-align-j .initial:n = \{}
,out-h .tl_set:N = \l_ducksay_output_h_pole_tl
,out-h .initial:n = l
,out-v .tl_set:N = \l_ducksay_output_v_pole_tl
,out-v .initial:n = vc
,out-x .dim_set:N = \l_ducksay_output_x_offset_dim
,out-x .value_required:n = true
,out-y .dim_set:N = \l_ducksay_output_y_offset_dim
,out-y .value_required:n = true
,t .meta:n = \{ out-v = t \}
,c .meta:n = \{ out-v = vc \}
,b .meta:n = \{ out-v = b \}
,\body* .tl_set:N = \l_ducksay_body_fount_tl
,\msg* .tl_set:N = \l_ducksay_msg_fount_tl
,bubble* .tl_set:N = \l_ducksay_bubble_fount_tl
,\body* .initial:n = \verbatim@font
,\msg* .initial:n = \verbatim@font
\end{Verbatim}
\end{verbatim}
Implementation of Version 2

\begin{verbatim}
,\texttt{\textbackslash body} .code:n = \tl_put_right:Nn \l_ducksay_body_fount_tl { \#1 }
,\texttt{\textbackslash msg} .code:n = \tl_put_right:Nn \l_ducksay_msg_fount_tl { \#1 }
,\texttt{\textbackslash bubble} .code:n = \tl_put_right:Nn \l_ducksay_bubble_fount_tl { \#1 }
,\texttt{\textbf{MSG}} .meta:n = { \texttt{msg} = \#1 , \texttt{bubble} = \#1 }
,\texttt{\textbf{MSG*}} .meta:n = { \texttt{msg*} = \#1 , \texttt{bubble*} = \#1 }
,\texttt{\textbackslash hpad} .int_set:N = \l_ducksay_hpad_int
,\texttt{\textbackslash hpad} .initial:n = 2
,\texttt{\textbackslash hpad} .value_required:n = true
,\texttt{\textbackslash vpad} .int_set:N = \l_ducksay_vpad_int
,\texttt{\textbackslash vpad} .value_required:n = true
,\texttt{\textbackslash col} .tl_set:N = \l_ducksay_msg_tabular_column_tl
,\texttt{\textbackslash bubble-top-kern} .tl_set:N = \l_ducksay_bubble_top_kern_tl
,\texttt{\textbackslash bubble-top-kern} .initial:n = { -.5\textbackslash ex }
,\texttt{\textbackslash bubble-top-kern} .value_required:n = true
,\texttt{\textbackslash bubble-bot-kern} .tl_set:N = \l_ducksay_bubble_bottom_kern_tl
,\texttt{\textbackslash bubble-bot-kern} .initial:n = { .2\textbackslash ex }
,\texttt{\textbackslash bubble-bot-kern} .value_required:n = true
,\texttt{\textbackslash bubble-side-kern} .tl_set:N = \l_ducksay_bubble_side_kern_tl
,\texttt{\textbackslash bubble-side-kern} .initial:n = { .2\textbackslash em }
,\texttt{\textbackslash bubble-side-kern} .value_required:n = true
,\texttt{\textbackslash bubble-delim-top} .tl_set:N = \l_ducksay_bubble_delim_top_tl
,\texttt{\textbackslash bubble-delim-left-1} .tl_set:N = \l_ducksay_bubble_delim_left_a_tl
,\texttt{\textbackslash bubble-delim-left-2} .tl_set:N = \l_ducksay_bubble_delim_left_b_tl
,\texttt{\textbackslash bubble-delim-left-3} .tl_set:N = \l_ducksay_bubble_delim_left_c_tl
,\texttt{\textbackslash bubble-delim-left-4} .tl_set:N = \l_ducksay_bubble_delim_left_d_tl
,\texttt{\textbackslash bubble-delim-right-1} .tl_set:N = \l_ducksay_bubble_delim_right_a_tl
,\texttt{\textbackslash bubble-delim-right-2} .tl_set:N = \l_ducksay_bubble_delim_right_b_tl
,\texttt{\textbackslash bubble-delim-right-3} .tl_set:N = \l_ducksay_bubble_delim_right_c_tl
,\texttt{\textbackslash bubble-delim-right-4} .tl_set:N = \l_ducksay_bubble_delim_right_d_tl
,\texttt{\textbackslash bubble-delim-top} .initial:n = { { - } }
,\texttt{\textbackslash bubble-delim-left-1} .initial:n = ( \\
,\texttt{\textbackslash bubble-delim-left-2} .initial:n = / \\
,\texttt{\textbackslash bubble-delim-left-3} .initial:n = | \\
,\texttt{\textbackslash bubble-delim-left-4} .initial:n = \texttt{\textbackslash c_backslash_str} \\
,\texttt{\textbackslash bubble-delim-right-1} .initial:n = ) \\
,\texttt{\textbackslash bubble-delim-right-2} .initial:n = \texttt{\textbackslash c_backslash_str} \\
,\texttt{\textbackslash bubble-delim-right-3} .initial:n = | \\
,\texttt{\textbackslash bubble-delim-right-4} .initial:n = / \\
,\texttt{\textbackslash strip-spaces} .bool_set:N = \l_ducksay_msg_strip_spaces_bool
\end{verbatim}

Redefine keys only intended for version 1 to throw an error:

\begin{verbatim}
\clist_map_inline:nn
{ align, rel-align }
\keys_define:nn { ducksay }
{ \texttt{\textbackslash msg} .code:n = \msg_error:nn { ducksay } { v1-key-only } }
\end{verbatim}

\subsection*{2.3.4 Functions}

\subsection*{2.3.4.1 Internal}

\verbatim
{26}
\end{verbatim}
Implementation of Version 2

\str_case:Vn \l_ducksay_msg_align_tl
{
  { l } \exp_not:N \l_ducksay_msg_align_l_tl
  { c } \exp_not:N \l_ducksay_msg_align_c_tl
  { r } \exp_not:N \l_ducksay_msg_align_r_tl
  { j } \exp_not:N \l_ducksay_msg_align_j_tl
}

(End definition for \ducksay_evaluate_message_alignment_fixed_width_common: This function is documented on page ??.)

\ducksay_evaluate_message_alignment_fixed_width_tabular:
\cs_new:Npn \ducksay_evaluate_message_alignment_fixed_width_tabular:
{
  \tl_if_empty:NT \l_ducksay_msg_tabular_column_tl
  \tl_set:Nx \l_ducksay_msg_tabular_column_tl
  {
    >
    { \ducksay_evaluate_message_alignment_fixed_width_common:
      \exp_not:N \arraybackslash
    }
    p \exp_not:N \l_ducksay_msg_width_dim
  }
}

(End definition for \ducksay_evaluate_message_alignment_fixed_width_tabular: This function is documented on page ??.)

\ducksay_evaluate_message_alignment_fixed_width_vbox:
\cs_new:Npn \ducksay_evaluate_message_alignment_fixed_width_vbox:
{
  \tl_set:Nx \l_ducksay_msg_align_vbox_tl
  { \ducksay_evaluate_message_alignment_fixed_width_common: }
}

(End definition for \ducksay_evaluate_message_alignment_fixed_width_vbox: This function is documented on page ??.)

\ducksay_calculate_msg_width_from_int:
\cs_new:Npn \ducksay_calculate_msg_width_from_int:
{
  \hbox_set:Nn \l_ducksay_tmpa_box \l_ducksay_msg_fount_tl M
  \dim_set:Nn \l_ducksay_msg_width_dim
  { \l_ducksay_msg_width_int \box_wd:N \l_ducksay_tmpa_box }
}

(End definition for \ducksay_calculate_msg_width_from_int: This function is documented on page ??.)
Implementation of Version 2

\str_case:Vn \l_ducksay_msg_align_tl
{
  { l }
  { \tl_set:Nn \l_ducksay_msg_tabular_column_tl { l } }
  { c }
  { \tl_set:Nn \l_ducksay_msg_tabular_column_tl { c } }
  { r }
  { \tl_set:Nn \l_ducksay_msg_tabular_column_tl { r } }
  { j }
  { \msg_error:nn { ducksay } { justify- unavailable } }
  { \tl_set:Nn \l_ducksay_msg_tabular_column_tl { l } }
}

\ducksay_calculate_msg_width_from_int:
\ducksay_evaluate_message_alignment_fixed_width_tabular:

\cs_set_eq:NN \ducksay_eat_argument:w \ducksay_eat_argument_tabular:w

(End definition for \ducksay_digest_options:n. This function is documented on page ??.)

\ducksay_set_bubble_top_kern:
\cs_new:Npn \ducksay_set_bubble_top_kern:
{
  \group_begin:
  \l_ducksay_bubble_fount_tl
  \exp_args:NNNx
  \group_end:
  \dim_set:Nn \l_ducksay_bubble_top_kern_dim
  { \dim_eval:n { \l_ducksay_bubble_top_kern_tl } }
}

(End definition for \ducksay_set_bubble_top_kern:. This function is documented on page ??.)

\ducksay_set_bubble_bottom_kern:
\cs_new:Npn \ducksay_set_bubble_bottom_kern:
{
  \group_begin:
  \l_ducksay_bubble_fount_tl
  \exp_args:NNNx
  \group_end:
  \dim_set:Nn \l_ducksay_bubble_bottom_kern_dim
  { \dim_eval:n { \l_ducksay_bubble_bottom_kern_tl } }
}

(End definition for \ducksay_set_bubble_bottom_kern:. This function is documented on page ??.)
\ducksay_shipout:

\cs_new_protected:Npn \ducksay_shipout:
  { \hcoffin_set:Nn \l_ducksay_msg_coffin { \box_use:N \l_ducksay_msg_box } \bool_if:FN \l_ducksay_no_bubble_bool
  \hbox_set:Nn \l_ducksay_tmpa_box
  { \l_ducksay_bubble_fount_tl \l_ducksay_bubble_delim_top_tl } \int_set:Nn \l_ducksay_msg_width_int
  \fp_eval:n
  \hbox_set:Nn \l_ducksay_tmpb_box
  \int_set:Nn \l_ducksay_msg_height_int
  \arraystretch * \baselineskip
  + \l_ducksay_vpad_int
  \begin{tabular}{@{}l@{}}
  \int_compare:nNnTF { \l_ducksay_msg_height_int } = { \c_one_int }
  \l_ducksay_bubble_fount_tl
  \begin{tabular}{@{}l@{}}
  \l_ducksay_bubble_delim_left_a_tl
  \l_ducksay_bubble_delim_left_b_tl\\
  \int_step_inline:nnn { 3 } { \l_ducksay_msg_height_int }
  \end{tabular}
  \end{tabular}
}
Implementation of Version 2

\kern-\l_ducksay_bubble_side_kern_tl
\l_ducksay_bubble_delim_left_c_tl
\l_ducksay_bubble_delim_left_d_tl
\end{tabular}
\hcoffin_set:Nn \l_ducksay_bubble_close_coffin
{ \l_ducksay_bubble_fount_tl
\begin{tabular}{@{}r@{}}
\int_compare:nNnTF { \l_ducksay_msg_height_int } = { \c_one_int }
{ \l_ducksay_bubble_delim_right_a_tl
{ \l_ducksay_bubble_delim_right_b_tl \l_ducksay_bubble_delim_right_c_tl\kern-\l_ducksay_bubble_side_kern_tl
\l_ducksay_bubble_delim_right_d_tl
}
\end{tabular}
}
\hcoffin_set:Nn \l_ducksay_bubble_top_coffin
{ \l_ducksay_bubble_fount_tl
\begin{tabular}{@{}r@{}}
\int_step_inline:nnn
{ 3 } { \l_ducksay_msg_height_int }
{ \l_ducksay_bubble_delim_right_c_tl\kern-\l_ducksay_bubble_side_kern_tl
\l_ducksay_bubble_delim_right_d_tl
}
\end{tabular}
}
\dim_set:Nn \l_ducksay_hpad_dim
{ (\coffin_wd:N \l_ducksay_bubble_top_coffin
- \coffin_wd:N \l_ducksay_msg_coffin ) / 2 }
\coffin_join:NnnNnnn
\l_ducksay_msg_coffin { l } { vc }
\l_ducksay_open_coffin { r } { vc }
- \l_ducksay_hpad_dim } \{ \c_zero_dim 
\coffin_join:NnnNnnn
\l_ducksay_msg_coffin { r } { vc }
\l_ducksay_close_coffin { l } { vc }
\l_ducksay_hpad_dim } \{ \c_zero_dim 
\ducksay_set_bubble_top_kern:
\ducksay_set_bubble_bottom_kern:
\coffin_join:NnnNnnn
(31)
\ducksay_bubble_close
\ducksay_bubble_open
2.3.4.1.1 Message Reading Functions  Version 2 has different ways of reading the message argument of \ducksay and \duckthink. They all should allow almost arbitrary content and the height and width are set based on the dimensions.
Implementation of Version 2

(End definition for `ducksay_eat_argument_tabular:w`. This function is documented on page ??.)

\ducksay_eat_argument_tabular_inner:w

\cs_new:Npn \ducksay_eat_argument_tabular_inner:w #1
\hbox_set:Nn \l_ducksay_msg_box
\{ \l_ducksay_msg_fount_tl \ducksay_msg_tabular_begin: #1 \ducksay_msg_tabular_end: \}
\ducksay_shipout:

(End definition for `ducksay_eat_argument_tabular_inner:w`. This function is documented on page ??.)

\ducksay_eat_argument_tabular_verb:w

\NewDocumentCommand \ducksay_eat_argument_tabular_verb:w { >{ \ducksay_process_verb_newline:nnn { ~ } { ~ \par } } +v }
\{ \ducksay_eat_argument_tabular_inner:w
\{ \group_begin:
\tex_everyeof:D { \exp_not:N }
\exp_after:wN
\group_end:
\tex_scantokens:D { #1 }
\}
\}

(End definition for `ducksay_eat_argument_tabular_verb:w`. This function is documented on page ??.)

\ducksay_eat_argument_tabular_normal:w

\NewDocumentCommand \ducksay_eat_argument_tabular_normal:w { +m }
\{ \ducksay_eat_argument_tabular_inner:w { #1 } \}

(End definition for `ducksay_eat_argument_tabular_normal:w`. This function is documented on page ??.)

\ducksay_eat_argument_hbox:w

\cs_new_protected_nopar:Npn \ducksay_eat_argument_hbox:w
\{ \bool_if:NTF \l_ducksay_msg_strip_spaces_bool
\{ \grabbox \}
\{ \grabbox* \}
\l_ducksay_msg_box \l_ducksay_msg_fount_tl \hbox \ducksay_shipout:
\}

(End definition for `ducksay_eat_argument_hbox:w`. This function is documented on page ??.)
Implementation of Version 2

\ducksay_eat_argument_vbox:w

\cs_new_protected_nopar:Npn \ducksay_eat_argument_vbox:w
\{
\ducksay_evaluate_message_alignment_fixed_width_vbox:
\bool_if:NTF \l_ducksay_msg_strip_spaces_bool
\{ \grabbox }
\{ \grabbox* }
\[
\hspace \l_ducksay_msg_width_dim
\linewidth \hspace
\l_ducksay_msg_fount_tl
\l_ducksay_msg_align_vbox_tl
\@afterindentfalse
\@afterheading
\]
\l_ducksay_msg_box
\vbox \ducksay_shipout:
\}

(End definition for \ducksay_eat_argument_vbox:w. This function is documented on page ??.)

2.3.4.1.2 Generating Variants of External Functions

\cs_generate_variant:Nn \coffin_join:NnnNnnnn { NVnNVnnn }\cs_generate_variant:Nn \coffin_attach:NnnNnnnn { NVnNVnnn }\cs_generate_variant:Nn \coffin_typeset:Nnnnn { NVVnn }\cs_generate_variant:Nn \str_case:nn { Vn }

2.3.4.2 Document level

\ducksay
\NewDocumentCommand \ducksay { O{} }\{ \ducksay_digest_options:n { #1 }\}

(End definition for \ducksay. This function is documented on page 7.)

\duckthink
\NewDocumentCommand \duckthink { O{} }\{ \ducksay_digest_options:n { think, #1 }\}

(End definition for \duckthink. This function is documented on page 7.)
2.4 Definition of the Animals

(*animals*)

%"A some of the below are from http://ascii.co.uk/art/kangaroo

\AddAnimal{duck} %

\AddAnimal{small-duck} %

\AddAnimal{duck-family} %

\AddAnimal{cow} %

\AddAnimal{head-in} %

\AddAnimal{sodomized} %

\AddAnimal{tux} %
```
```
\AddAnimal\{tail-count=1\}\{crusader\}\%>>>

\csname bool_if:cT\endcsname \{l_ducksay_version_one_bool\}
\AnimalOptions\{crusader\}\{tail-1=|,rel-align=c\}

%^^A http://ascii.co.uk/art/knights
\AddAnimal\{tail-count=3\}\{knight\}\%>>>

⟨/animals⟩
Who’s gonna use it anyway?

Hosted at https://github.com/Skillmon/ltx_ducksay it is.