

\

 >(' ')

)/

 /(

 / '-----/

 \ ~=- /

 ~~~~~

>()\_  
( )\_ \_

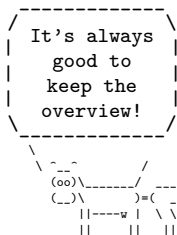
( by Jonathan P. Spratte )

( Today is 2019-06-11 )

```

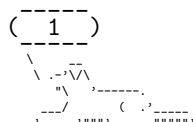
\      .\|//|\| |.
\    |/\|/|/|//|/|
    /. ' |/\|/|/|/|
    o_-,_|//|/|/\| |'

```



# Contents

|          |                                           |           |
|----------|-------------------------------------------|-----------|
| <b>1</b> | <b>Documentation</b>                      | <b>2</b>  |
| 1.1      | Downward Compatibility Issues             | 2         |
| 1.2      | Shared between versions                   | 3         |
| 1.2.1    | Macros                                    | 3         |
| 1.2.2    | Options                                   | 4         |
| 1.2.2.1  | Options for <code>\AddAnimal</code>       | 5         |
| 1.3      | Version 1                                 | 6         |
| 1.3.1    | Introductkion                             | 6         |
| 1.3.2    | Macros                                    | 6         |
| 1.3.3    | Options                                   | 6         |
| 1.3.4    | Defects                                   | 7         |
| 1.4      | Version 2                                 | 8         |
| 1.4.1    | Introductkion                             | 8         |
| 1.4.2    | Macros                                    | 8         |
| 1.4.3    | Options                                   | 8         |
| 1.5      | Dependencies                              | 14        |
| 1.6      | Available Animals                         | 14        |
| 1.7      | License and Bug Reports                   | 16        |
| <b>2</b> | <b>Implementation</b>                     | <b>17</b> |
| 2.1      | Shared between versions                   | 17        |
| 2.1.1    | Variables                                 | 17        |
| 2.1.1.1  | Integers                                  | 17        |
| 2.1.1.2  | Sequences                                 | 17        |
| 2.1.1.3  | Token lists                               | 17        |
| 2.1.1.4  | Boolean                                   | 17        |
| 2.1.1.5  | Boxes                                     | 17        |
| 2.1.2    | Regular Expressions                       | 17        |
| 2.1.3    | Messages                                  | 17        |
| 2.1.4    | Key-value setup                           | 17        |
| 2.1.4.1  | Keys for <code>\AddAnimal</code>          | 18        |
| 2.1.5    | Functions                                 | 19        |
| 2.1.5.1  | Generating Variants of External Functions | 19        |
| 2.1.5.2  | Internal                                  | 19        |
| 2.1.5.3  | Document level                            | 21        |
| 2.1.6    | Load the Correct Version and the Animals  | 22        |
| 2.2      | Version 1                                 | 23        |
| 2.2.1    | Functions                                 | 23        |
| 2.2.1.1  | Internal                                  | 23        |
| 2.2.1.2  | Document level                            | 25        |
| 2.3      | Version 2                                 | 26        |
| 2.3.1    | Messages                                  | 26        |
| 2.3.2    | Variables                                 | 26        |
| 2.3.2.1  | Token Lists                               | 26        |
| 2.3.2.2  | Boxes                                     | 26        |
| 2.3.2.3  | Bools                                     | 26        |
| 2.3.2.4  | Coffins                                   | 26        |
| 2.3.2.5  | Dimensions                                | 26        |



|           |                                           |    |
|-----------|-------------------------------------------|----|
| 2.3.3     | Options                                   | 26 |
| 2.3.4     | Functions                                 | 29 |
| 2.3.4.1   | Internal                                  | 29 |
| 2.3.4.1.1 | Message Reading Functions                 | 35 |
| 2.3.4.1.2 | Generating Variants of External Functions | 36 |
| 2.3.4.2   | Document level                            | 36 |
| 2.4       | Definition of the Animals                 | 38 |

Just beest mod'rn,  
thee peasant!



## 1 Documentation

This is ducksay! A cowsay for L<sup>A</sup>T<sub>E</sub>X. ducksay is part of T<sub>E</sub>XLive and MiK<sub>T</sub>E<sub>X</sub> since September 2017. If it is not part of your installation it means that your L<sup>A</sup>T<sub>E</sub>X installation is *really* out of date, you have two options: Update your installation or try to install ducksay yourself. Chances are that if you opt for the latter, the version of expl3 in your L<sup>A</sup>T<sub>E</sub>X installation is too old, too, and the l3regex module is not yet part of expl3. In that case you'll get a few undefined control sequence errors. `\usepackage{l3regex}` prior to loading ducksay might fix these issues. Additionally you'll need grabbox for version 2 of ducksay that won't be part of your L<sup>A</sup>T<sub>E</sub>X installation, too. Please note that I don't actively support out of date L<sup>A</sup>T<sub>E</sub>X installations, so if loading l3regex doesn't fix the issues and you're on an old installation, I won't provide further support.

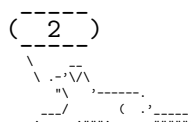
Yep, I screwed up!



### 1.1 Downward Compatibility Issues

In the following list I use the term “version” to refer to package versions, the same is true if I use an abbreviation like “v2.0” (or anything that matches the regular expression `v\d+(\.d+)?`). For the code variant which can be set using the `version` option I'll use the term “variant” or specify directly that I'm referring to that option (the used font may be a hint, too).

- v2.0
  - Versions prior to v2.0 did use a regular expression for the option `ligatures`, see [subsection 1.2.2](#) for more on this issue.
  - In a document created with package versions prior to v2.0 you'll have to specify the option `version=1` with newer package 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 [subsection 1.2.1](#) for more about the current behaviour.
- v2.4
  - The `add-think` key was deprecated in v2.3 and was removed in v2.4 since the output symbols of the bubble tail are handled differently and more efficient now.



```

-----
\ Macros for everyone!
/
\ ^
 \
  ( )
   ( )
  .( )

```

## 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 [subsection 1.2.2](#) for the code variants, [subsection 1.3.2](#) and [subsection 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 [subsection 1.2.2](#), [subsection 1.3.3](#) and [subsection 1.4.3](#). Don't use an `<animal>` here, it has no effect.

---

`\AddAnimal`


---

`\AddAnimal{<*>[<options>]{<animal>}<ascii-art>}`

adds `<animal>` to the known animals. `<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 `<animal>` is the new default. One space is added to the begin of `<animal>` (compensating the opening symbol). The symbols signaling the speech bubble's tail (in the `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, `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{<*>[<options>]{<animal>}<ascii-art>}`

It does the same as `\AddAnimal` but allows three different colouring syntaxes. You can use `\textcolor` in the `<ascii-art>` with the syntax `\textcolor{<color>}{<text>}`. Note that you can't use braces in the arguments of `\textcolor`.

You can also use a delimited `\color` of the form `\bgroup\color{<color>}<text>\egroup`, a space after that `\egroup` will be considered a space in the output, so you don't have to care for correct termination of the `\egroup` (so `\bgroup\color{red}RedText\egroupOtherText` is valid syntax). You can't nest delimited `\colors`.

Also you can use an undelimited `\color`. It affects anything until the end of the current line (or, if used inside of the `<text>` of a delimited `\color`, anything until the end of that delimited `\color`'s `<text>`). The syntax would be `\color{<color>}`.

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 L<sup>A</sup>T<sub>E</sub>X3. It is therefore slower than the normal `\AddAnimal`.

```

( 3 )
\ .-' \
  " \
  / ( '
  /-----\

```

---

`\AnimalOptions` `\AnimalOptions{*}{animal}{options}`

---

With this macro you can set *animal* specific *options*. If the star is given any currently set options for this *animal* are dropped and only the ones specified in *options* will be applied, else *options* will be added to the set options for this *animal*. The set *options* can set the `tail-1` and `tail-2` options and therefore overwrite the effects of `\duckthink`, as `\duckthink` really is just `\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 [subsection 1.3.3](#) and [subsection 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`.

*animal* One of the animals listed in [subsection 1.6](#) or any of the ones added with `\AddAnimal`. Not useable as package option. Also don’t use it in `\DucksayOptions`, it’ll break the default animal selection.

`animal=<animal>`

Locally sets the default animal. Note that `\ducksay` and `\duckthink` do digest their options inside of a group, so it just results in a longer alternative to the use of *animal* if used in their options.

`ligatures=<token list>`

each token you don’t want to form ligatures during `\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 `\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 `<>,-`.

**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).

`no-tail` Sets `tail-1` and `tail-2` to be a space.

`say` Sets `tail-1` and `tail-2` as backslashes.

`tail-1=<token list>`

Sets the first tail symbol in the output to be *token list*. If set outside of `\ducksay` and `\duckthink` it will be overwritten inside of `\duckthink` to be `0`.

```
Options.
For every occasion
\_/oo } }-@
('')_ } |
'--' | { }--{ }
//_ /_ /
```

```
( 4 )
\ .-'\ \
" \
_/_ ( '-----
)-----)-----)-----)
```

$$\text{tail-2} = \langle \text{token list} \rangle$$

Sets every other tail symbol except the first one in the output to be `\token list`. If set outside of `\ducksay` and `\duckthink` it will be overwritten inside of `\duckthink` to be `o`.

think       Sets tail-1=0 and tail-2=0.

### 1.2.2.1 Options for \AddAnimal

The options described here are only available in `\AddAnimal` and `\AddColoredAnimal`.

```
tail-count=<int>
```

sets the number of tail symbols to be replaced in `\AddAnimal` and `\AddColoredAnimal`. Initial value is 2. If the value is negative every occurrence of `tail-symbol` will be replaced.

tail-symbol=*<str>*

the symbol used in `\AddAnimal` and `\AddColoredAnimal` to mark the bubble's tail. The argument gets `\detokenized`. Initially a single backslash.

( 5 )

## 1.3 Version 1

### 1.3.1 Introducktion

This version is included for legacy support (old documents should behave the same without any change to them – except the usage of `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 `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.

`\ducksay[⟨options⟩]{⟨message⟩}`

options might include any of the options described in [subsection 1.2.2](#) and [subsection 1.3.3](#) if not otherwise specified. Prints an *⟨animal⟩* saying *⟨message⟩*. *⟨message⟩* is not read in verbatim. Multi-line *⟨message⟩*s are possible using `\\`. `\\` should not be contained in a macro definition but at toplevel. Else use the option `ht`.

`\duckthink[⟨options⟩]{⟨message⟩}`

options might include any of the options described in [subsection 1.2.2](#) and [subsection 1.3.3](#) if not otherwise specified. Prints an *⟨animal⟩* thinking *⟨message⟩*. *⟨message⟩* is not read in verbatim. Multi-line *⟨message⟩*s are possible using `\\`. `\\` should not be contained in a macro definition but at toplevel. Else use the option `ht`.

### 1.3.3 Options

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

`bubble=⟨code⟩`

use *⟨code⟩* 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.

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

`align=⟨valign⟩`

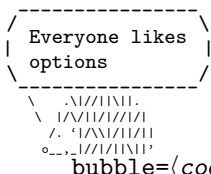
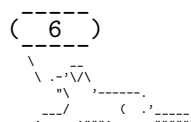
use *⟨valign⟩* as the vertical alignment specifier given to the `tabular` which is around the contents of `\ducksay` and `\duckthink`.

`msg-align=⟨halign⟩`

use *⟨halign⟩* for alignment of the rows of multi-line *⟨message⟩*s. It should match a `tabular` column specifier. Default is 1. It only affects the contents of the speech bubble not the bubble.

`rel-align=⟨column⟩`

use *⟨column⟩* for alignment of the bubble and the body. It should match a `tabular` column specifier. Default is 1.



`wd=<count>` in order to detect the width the `<message>` is expanded. This might not work out for some commands (e.g. `\url` from `hyperref`). If you specify the width using `wd` the `<message>` is not expanded and therefore the command *might* work out. `<count>` should be the character count.

`ht=<count>` you might explicitly set the height (the row count) of the `<message>`. This only has an effect if you also specify `wd`.

### 1.3.4 Defects

```

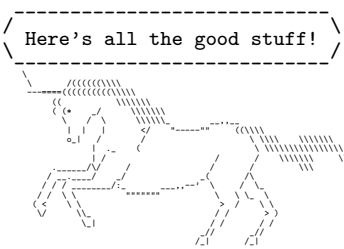
      /-----\
    /  Ohh, no!  \
    \-----/
      \
       \  (.)_(.)
        \  (    )  _
         / \ '-----' \ \
        --\ ( (    ) ) /--
           )  / \ \_./ \ / (
            )_/ / \ \  / \ \_ (

```

- no automatic line wrapping
- message width detection based on token count with `\edef` expansion, might fail badly

( 7 )





Here's all the good stuff!

## 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` `\ducksay[⟨options⟩]{⟨message⟩}`

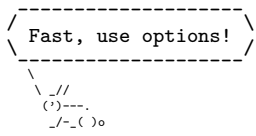
options might include any of the options described in [subsection 1.2.2](#) and [subsection 1.4.3](#) if not otherwise specified. Prints an `⟨animal⟩` saying `⟨message⟩`.

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 [subsection 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 L<sup>A</sup>T<sub>E</sub>X3's coffin mechanism described in [interface3.pdf](#) and the documentation of `xcoffins`.

`\duckthink` `\duckthink[⟨options⟩]{⟨message⟩}`

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



Fast, use options!

### 1.4.3 Options

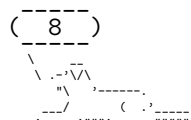
In version 2 the following options are available. Keep in mind that you shouldn't use them during package load time but in the arguments of `\ducksay`, `\duckthink` or `\DucksayOptions`.

`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`.



( 8 )

**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.

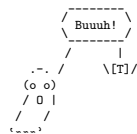
**b** shortcut for `out-v=b`.

`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-bigger=<count>`  
vertically enlarge the body by `<count>` empty lines added to the bottom. This way top-aligning two different body types is easier (by actually bottom aligning the two):



```
\ducksay[ghost,body-x=-7mm,b,body-mirrored]{Buuuh!}
\ducksay[crusader,body-bigger=4,b,out-h=r,no-bubble]{}
```

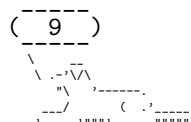
`body-mirrored=<bool>`  
if set true the `<animal>` will be mirrored along its vertical centre axis. Package default is `false`. If you set it `true` you'll most likely need to manually adjust the alignment of the body with one or more of the keys `body-align`, `body-to-msg`, `msg-to-body`, `body-x` 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.
- `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`.

( 11 )

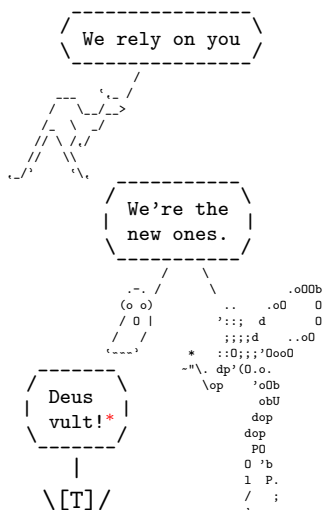
- `msg-align-r=<token list>`  
 set the `<token list>` which is responsible to typeset the message flushed right if the option `msg-align=r` is used. It is used independent of the `arg` key. For `arg=tab` and `arg=tab*` it is only used if a fixed width is specified and the macro `\arraybackslash` provided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

```

( 12 )
-----
 \ .-'√\
  " \   '-----
   /   ( .')-----
  /-----) HHHH)-----) HHHHHH)

```

`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.

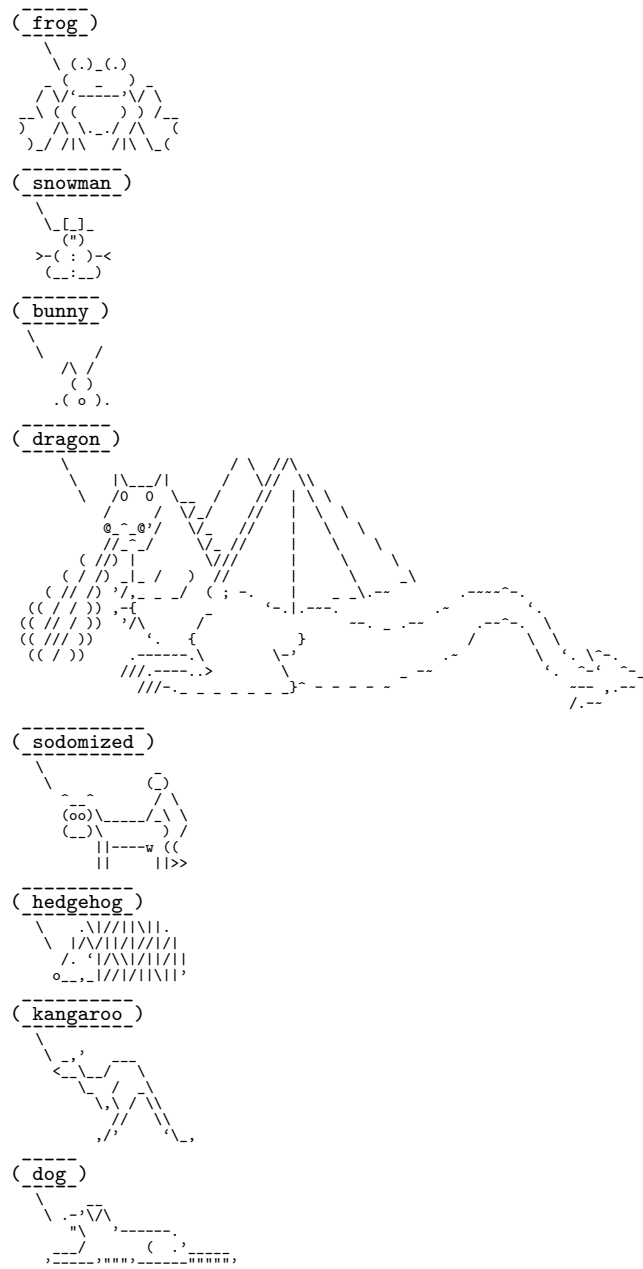
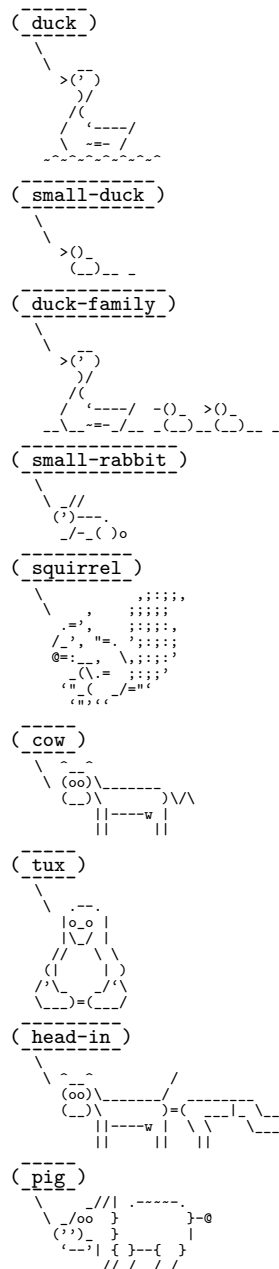


## 1.5 Dependencies

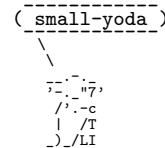
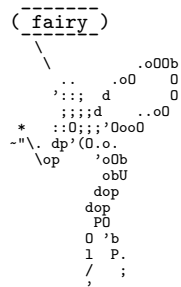
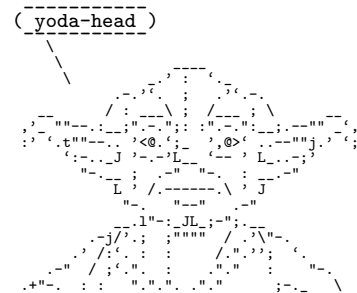
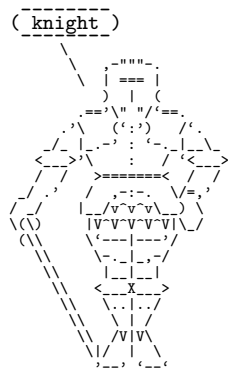
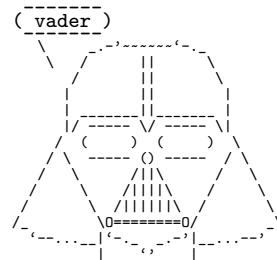
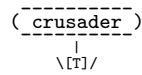
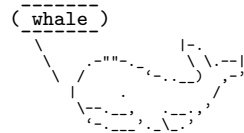
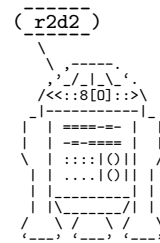
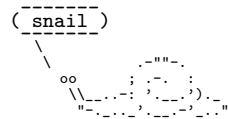
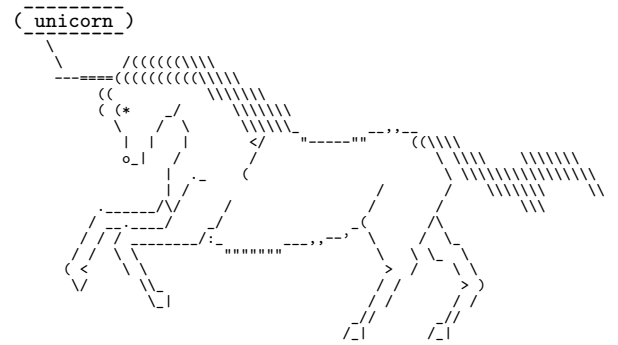
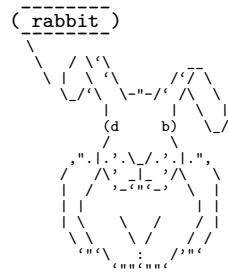
The package depends on the two packages `xparse` and `l3keys2e` and all of their dependencies. Version 2 additionally depends on `array` and `grabbbox`.

## 1.6 Available Animals

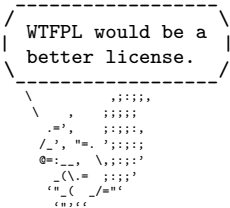
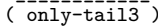
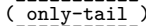
The following animals are provided by this package. I did not create them (but altered some), they belong to their original creators.



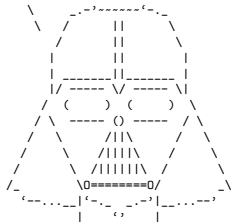
\*Latin; "I feel fusty already."







Only rebel scum reads  
documentation!  
Join the dark side,  
read the implementation.



## 2 Implementation

1 `<*pkg>`

### 2.1 Shared between versions

#### 2.1.1 Variables

##### 2.1.1.1 Integers

2 `\int_new:N \l_ducksay_msg_width_int`  
3 `\int_new:N \l_ducksay_msg_height_int`  
4 `\int_new:N \l_ducksay_tail_symbol_count_int`

##### 2.1.1.2 Sequences

5 `\seq_new:N \l_ducksay_msg_lines_seq`

##### 2.1.1.3 Token lists

6 `\tl_new:N \l_ducksay_align_tl`  
7 `\tl_new:N \l_ducksay_msg_align_tl`  
8 `\tl_new:N \l_ducksay_animal_tl`  
9 `\tl_new:N \l_ducksay_body_tl`  
10 `\tl_new:N \l_ducksay_bubble_tl`  
11 `\tl_new:N \l_ducksay_tmpa_tl`  
12 `\tl_new:N \l_ducksay_tail_symbol_out_one_tl`  
13 `\tl_new:N \l_ducksay_tail_symbol_out_two_tl`  
14 `\tl_new:N \l_ducksay_tail_symbol_in_tl`

##### 2.1.1.4 Boolean

15 `\bool_new:N \l_ducksay_version_one_bool`  
16 `\bool_new:N \l_ducksay_version_two_bool`

##### 2.1.1.5 Boxes

17 `\box_new:N \l_ducksay_tmpa_box`

### 2.1.2 Regular Expressions

Regular expressions for `\AddColoredAnimal`

18 `\regex_const:Nn \c_ducksay_textcolor_regex`  
19 `{ \c0(?:\\textcolor\{(.*)\}\{(.*)\}) }`  
20 `\regex_const:Nn \c_ducksay_color_delim_regex`  
21 `{ \c0(?:\\bgroup\\color\{(.*)\}(.*\\egroup) }`  
22 `\regex_const:Nn \c_ducksay_color_regex`  
23 `{ \c0(?:\\color\{(.*)\}) }`

### 2.1.3 Messages

24 `\msg_new:nnn { ducksay } { load-time-only }`  
25 `{ The~'#1'~key-is-to-be-used-only~during~package~load~time. }`

### 2.1.4 Key-value setup

26 `\keys_define:nn { ducksay }`  
27 `{`  
28  `,bubble .tl_set:N = \l_ducksay_bubble_tl`  
29  `,body .tl_set:N = \l_ducksay_body_tl`  
30  `,align .tl_set:N = \l_ducksay_align_tl`  
31  `,align .value_required:n = true`

( 17 )  
\\ .-'\V\-----  
" \'  
----- ( . )-----  
)-----)-----)-----)

```

32 ,wd .int_set:N = \l_ducksay_msg_width_int
33 ,wd .initial:n = -\c_max_int
34 ,wd .value_required:n = true
35 ,ht .int_set:N = \l_ducksay_msg_height_int
36 ,ht .initial:n = -\c_max_int
37 ,ht .value_required:n = true
38 ,animal .code:n =
39 { \keys_define:nn { ducksay } { default_animal .meta:n = { #1 } } }
40 ,animal .initial:n = duck
41 ,msg-align .tl_set:N = \l_ducksay_msg_align_tl
42 ,msg-align .initial:n = 1
43 ,msg-align .value_required:n = true
44 ,rel-align .tl_set:N = \l_ducksay_rel_align_tl
45 ,rel-align .initial:n = 1
46 ,rel-align .value_required:n = true
47 ,ligatures .tl_set:N = \l_ducksay_ligatures_tl
48 ,ligatures .initial:n = { '<>','-' }
49 ,tail-1 .tl_set:N = \l_ducksay_tail_symbol_out_one_tl
50 ,tail-1 .initial:x = \c_backslash_str
51 ,tail-2 .tl_set:N = \l_ducksay_tail_symbol_out_two_tl
52 ,tail-2 .initial:x = \c_backslash_str
53 ,no-tail .meta:n = { tail-1 = { ~ }, tail-2 = { ~ } }
54 ,think .meta:n = { tail-1 = { 0 }, tail-2 = { o } }
55 ,say .code:n =
56 {
57 \exp_args:Nx \DucksayOptions
58 { tail-1 = { \c_backslash_str }, tail-2 = { \c_backslash_str } }
59 }
60 ,version .choice:
61 ,version / 1 .code:n =
62 {
63 \bool_set_false:N \l_ducksay_version_two_bool
64 \bool_set_true:N \l_ducksay_version_one_bool
65 }
66 ,version / 2 .code:n =
67 {
68 \bool_set_false:N \l_ducksay_version_one_bool
69 \bool_set_true:N \l_ducksay_version_two_bool
70 }
71 ,version .initial:n = 2
72 }
73 \ProcessKeysOptions { ducksay }
74 \keys_define:nn { ducksay }
75 {
76 version .code:n = \msg_error:nnn { ducksay } { load-time-only } { version }
77 }

```

#### 2.1.4.1 Keys for \AddAnimal

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

```

78 \keys_define:nn { ducksay / add-animal }
79 {
80 ,tail-symbol .code:n =

```

```

( 18 )
\ .-'\
" \
----- ( .)-----
)-----)-----)-----)-----)

```

```

81     \tl_set:Nx \l_ducksay_tail_symbol_in_tl { \tl_to_str:n { #1 } }
82     ,tail-symbol .initial:o = \c_backslash_str
83     ,tail-count .int_set:N = \l_ducksay_tail_symbol_count_int
84     ,tail-count .initial:n = 2
85 }

```

## 2.1.5 Functions

### 2.1.5.1 Generating Variants of External Functions

```

86 \cs_generate_variant:Nn \tl_replace_once:Nnn { NVn }
87 \cs_generate_variant:Nn \tl_replace_all:Nnn { NVn }

```

### 2.1.5.2 Internal

`\__ducksay_everyeof:w`

```

88 \cs_set_eq:NN \__ducksay_everyeof:w \tex_everyeof:D

```

*(End definition for \\_\_ducksay\_everyeof:w.)*

`\__ducksay_scantokens:w`

```

89 \cs_set_eq:NN \__ducksay_scantokens:w \tex_scantokens:D

```

*(End definition for \\_\_ducksay\_scantokens:w.)*

`\ducksay_replace_verb_newline:Nn`

```

90 \cs_new_protected:Npx \ducksay_replace_verb_newline:Nn #1 #2
91 {
92     \tl_replace_all:Nnn #1 { \char_generate:nn { 13 } { 12 } } { #2 }
93 }

```

*(End definition for \ducksay\_replace\_verb\_newline:Nn. This function is documented on page ??.)*

`\ducksay_replace_verb_newline_newline:Nn`

```

94 \cs_new_protected:Npx \ducksay_replace_verb_newline_newline:Nn #1 #2
95 {
96     \tl_replace_all:Nnn #1
97     { \char_generate:nn { 13 } { 12 } \char_generate:nn { 13 } { 12 } } { #2 }
98 }

```

*(End definition for \ducksay\_replace\_verb\_newline\_newline:Nn. This function is documented on page ??.)*

`\ducksay_process_verb_newline:nnn`

```

99 \cs_new_protected:Npn \ducksay_process_verb_newline:nnn #1 #2 #3
100 {
101     \tl_set:Nn \ProcessedArgument { #3 }
102     \ducksay_replace_verb_newline_newline:Nn \ProcessedArgument { #2 }
103     \ducksay_replace_verb_newline:Nn \ProcessedArgument { #1 }
104 }

```

*(End definition for \ducksay\_process\_verb\_newline:nnn. This function is documented on page ??.)*

\ducksay\_add\_animal\_inner:nnnn

```
105 \cs_new_protected:Npn \ducksay_add_animal_inner:nnnn #1 #2 #3 #4
106 {
107   \group_begin:
108     \keys_set:nn { ducksay / add-animal } { #1 }
109     \tl_set:Nn \l_ducksay_tmpa_tl { \ #3 }
110     \int_compare:nNnTF { \l_ducksay_tail_symbol_count_int } < { \c_zero_int }
111     {
112       \tl_replace_once:NVn
113         \l_ducksay_tmpa_tl
114         \l_ducksay_tail_symbol_in_tl
115         \l_ducksay_tail_symbol_out_one_tl
116       \tl_replace_all:NVn
117         \l_ducksay_tmpa_tl
118         \l_ducksay_tail_symbol_in_tl
119         \l_ducksay_tail_symbol_out_two_tl
120     }
121     {
122       \int_compare:nNnT { \l_ducksay_tail_symbol_count_int } >
123       { \c_zero_int }
124       {
125         \tl_replace_once:NVn
126           \l_ducksay_tmpa_tl
127           \l_ducksay_tail_symbol_in_tl
128           \l_ducksay_tail_symbol_out_one_tl
129         \int_step_inline:nnn { 2 } { \l_ducksay_tail_symbol_count_int }
130         {
131           \tl_replace_once:NVn
132             \l_ducksay_tmpa_tl
133             \l_ducksay_tail_symbol_in_tl
134             \l_ducksay_tail_symbol_out_two_tl
135         }
136       }
137     }
138     \tl_map_inline:Nn \l_ducksay_ligatures_tl
139     { \tl_replace_all:Nnn \l_ducksay_tmpa_tl { ##1 } { { ##1 } } }
140     \ducksay_replace_verb_newline:Nn \l_ducksay_tmpa_tl
141     { \tabularnewline\null }
142     \exp_args:NNnV
143   \group_end:
144   \tl_set:cn { l_ducksay_animal_#2_tl } \l_ducksay_tmpa_tl
145   \exp_args:Nnx \keys_define:nn { ducksay }
146   {
147     #2 .code:n =
148     {
149       \exp_not:n { \tl_set_eq:NN \l_ducksay_animal_tl }
150       \exp_after:wN \exp_not:N \cs:w l_ducksay_animal_#2_tl \cs_end:
151       \exp_not:n { \exp_args:NV \DucksayOptions }
152       \exp_after:wN
153       \exp_not:N \cs:w l_ducksay_animal_#2_options_tl \cs_end:
154     }
155   }
156   \tl_if_exist:cF { l_ducksay_animal_#2_options_tl }
157   { \tl_new:c { l_ducksay_animal_#2_options_tl } }
```

```

158 \IfBooleanT { #4 }
159 { \keys_define:nn { ducksay } { default_animal .meta:n = { #2 } } }
160 }
161 \cs_generate_variant:Nn \ducksay_add_animal_inner:nnnn { nnVn }

```

(End definition for \ducksay\_add\_animal\_inner:nnnn. This function is documented on page ??.)

### 2.1.5.3 Document level

#### \DefaultAnimal

```

162 \NewDocumentCommand \DefaultAnimal { m }
163 {
164 \keys_define:nn { ducksay } { default_animal .meta:n = { #1 } }
165 }

```

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

#### \DucksayOptions

```

166 \NewDocumentCommand \DucksayOptions { m }
167 {
168 \keys_set:nn { ducksay } { #1 }
169 }

```

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

#### \AddAnimal

```

170 \NewDocumentCommand \AddAnimal { s O{} m +v }
171 {
172 \ducksay_add_animal_inner:nnnn { #2 } { #3 } { #4 } { #1 }
173 }

```

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

#### \AddColoredAnimal

```

174 \NewDocumentCommand \AddColoredAnimal { s O{} m +v }
175 {
176 \tl_set:Nn \l_ducksay_tmpa_tl { #4 }
177 \regex_replace_all:NnN \c_ducksay_color_delim_regex
178 { \c{bgroup}\c{color}\cB{\1\cE}\2\c{egroup} }
179 \l_ducksay_tmpa_tl
180 \regex_replace_all:NnN \c_ducksay_color_regex
181 { \c{color}\cB{\1\cE}\ }
182 \l_ducksay_tmpa_tl
183 \regex_replace_all:NnN \c_ducksay_textcolor_regex
184 { \c{textcolor}\cB{\1\cE}\cB{\2\cE}\ }
185 \l_ducksay_tmpa_tl
186 \ducksay_add_animal_inner:nnVn { #2 } { #3 } \l_ducksay_tmpa_tl { #1 }
187 }

```

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

## `\AnimalOptions`

```
188 \NewDocumentCommand \AnimalOptions { s m m }
189 {
190   \tl_if_exist:cTF { l_ducksay_animal_#2_options_tl }
191   {
192     \IfBooleanTF { #1 }
193     { \tl_set:cn }
194     { \tl_put_right:cn }
195   }
196   { \tl_set:cn }
197   { l_ducksay_animal_#2_options_tl } { #3, }
198 }
```

(End definition for `\AnimalOptions`. This function is documented on page 4.)

### 2.1.6 Load the Correct Version and the Animals

```
199 \bool_if:NT \l_ducksay_version_one_bool
200 { \file_input:n { ducksay.code.v1.tex } }
201 \bool_if:NT \l_ducksay_version_two_bool
202 { \file_input:n { ducksay.code.v2.tex } }
203 \ExplSyntaxOff
204 \input{ducksay.animals.tex}
205 </pkg>
```

## 2.2 Version 1

```

206 <*code.v1>
207 \ProvidesFile{ducksay.code.v1.tex}
208 [\ducksay@date\space v\ducksay@version\space ducksay code version 1]

```

### 2.2.1 Functions

#### 2.2.1.1 Internal

`\ducksay_longest_line:n` Calculate the length of the longest line

```

209 \cs_new:Npn \ducksay_longest_line:n #1
210 {
211   \int_incr:N \l_ducksay_msg_height_int
212   \exp_args:NNx \tl_set:Nn \l_ducksay_tmpa_tl { #1 }
213   \regex_replace_all:nnN { \s } { \c { space } } \l_ducksay_tmpa_tl
214   \int_set:Nn \l_ducksay_msg_width_int
215   {
216     \int_max:nn
217     { \l_ducksay_msg_width_int } { \tl_count:N \l_ducksay_tmpa_tl }
218   }
219 }

```

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

`\ducksay_open_bubble:` Draw the opening bracket of the bubble

```

220 \cs_new:Npn \ducksay_open_bubble:
221 {
222   \begin{tabular}{@{}l@{}}
223     \null\
224     \int_compare:nNnTF { \l_ducksay_msg_height_int } = { 1 } { ( }
225     {
226       /
227       \int_step_inline:nnn
228       { 3 } { \l_ducksay_msg_height_int } { \\kern-0.2em| }
229       \\detokenize{ \ }
230     }
231     \\[-1ex]\null
232   \end{tabular}
233   \begin{tabular}{@{}l@{}}
234     _\\
235     \int_step_inline:nnn { 2 } { \l_ducksay_msg_height_int } { \\ } \\[-1ex]
236     \mbox{ - }
237   \end{tabular}
238 }

```

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

`\ducksay_close_bubble:` Draw the closing bracket of the bubble

```

239 \cs_new:Npn \ducksay_close_bubble:
240 {
241   \begin{tabular}{@{}l@{}}
242     _\\
243     \int_step_inline:nnn { 2 } { \l_ducksay_msg_height_int } { \\ } \\[-1ex]
244     { - }
245   \end{tabular}

```

```

( 23 )
\ .-'\
" \
----- ( .)-----
)-----)

```



```

246 \begin{tabular}{@{}r@{}}
247 \null\\
248 \int_compare:nNnTF { \l_ducksay_msg_height_int } = { 1 }
249 { ) }
250 {
251 \detokenize {\ }
252 \int_step_inline:nnn
253 { 3 } { \l_ducksay_msg_height_int } { \\|\kern-0.2em }
254 \\/
255 }
256 \\[-1ex]\null
257 \end{tabular}
258 }

```

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

`\ducksay_print_msg:nn` Print out the message

```

259 \cs_new:Npn \ducksay_print_msg:nn #1 #2
260 {
261 \begin{tabular}{@{} #2 @{}}
262 \int_step_inline:nn { \l_ducksay_msg_width_int } { _ } \\
263 #1\\[-1ex]
264 \int_step_inline:nn { \l_ducksay_msg_width_int } { { - } }
265 \end{tabular}
266 }
267 \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

```

268 \cs_new:Npn \ducksay_print:nn #1 #2
269 {
270 \int_compare:nNnTF { \l_ducksay_msg_width_int } < { 0 }
271 {
272 \int_zero:N \l_ducksay_msg_height_int
273 \seq_set_split:Nnn \l_ducksay_msg_lines_seq { \\ } { #1 }
274 \seq_map_function:NN \l_ducksay_msg_lines_seq \ducksay_longest_line:n
275 }
276 {
277 \int_compare:nNnT { \l_ducksay_msg_height_int } < { 0 }
278 {
279 \regex_count:nnN { \c { \\ } } { #1 } \l_ducksay_msg_height_int
280 \int_incr:N \l_ducksay_msg_height_int
281 }
282 }
283 \group_begin:
284 \frenchspacing
285 \verbatim@font
286 \@noligs
287 \begin{tabular}[\l_ducksay_align_tl]{@{}#2@{}}
288 \l_ducksay_bubble_tl
289 \begin{tabular}{@{}l@{}}
290 \ducksay_open_bubble:
291 \ducksay_print_msg:nV { #1 } \l_ducksay_msg_align_tl
292 \ducksay_close_bubble:

```

```

( 24 )
\ .-'\ \
" \
----- ( .) -----
) -----)

```

```

293         \end{tabular}\\
294         \l_ducksay_body_tl
295         \begin{tabular}{@{}l@{}}
296             \l_ducksay_animal_tl
297         \end{tabular}
298     \end{tabular}
299 \group_end:
300 }
301 \cs_generate_variant:Nn \ducksay_print:nn { nV }

```

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

`\ducksay_say_and_think:nn` Reset some variables

```

302 \cs_new:Npn \ducksay_say_and_think:nn #1 #2
303 {
304     \group_begin:
305     \int_set:Nn \l_ducksay_msg_width_int { -\c_max_int }
306     \int_set:Nn \l_ducksay_msg_height_int { -\c_max_int }
307     \keys_set:nn { ducksay } { #1 }
308     \tl_if_empty:NT \l_ducksay_animal_tl
309     { \keys_set:nn { ducksay } { default_animal } }
310     \ducksay_print:nV { #2 } \l_ducksay_rel_align_tl
311 \group_end:
312 }

```

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

### 2.2.1.2 Document level

`\ducksay`

```

313 \NewDocumentCommand \ducksay { 0{} m }
314 {
315     \ducksay_say_and_think:nn { #1 } { #2 }
316 }

```

(End definition for `\ducksay`. This function is documented on page 8.)

`\duckthink`

```

317 \NewDocumentCommand \duckthink { 0{} m }
318 {
319     \ducksay_say_and_think:nn { think, #1 } { #2 }
320 }

```

(End definition for `\duckthink`. This function is documented on page 8.)

```

321 \</code.v1>

```

## 2.3 Version 2

```

322 <*code.v2>
323 \ProvidesFile{ducksay.code.v2.tex}
324 [\ducksay@date\space v\ducksay@version\space ducksay code version 2]

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

```

### 2.3.1 Messages

```

326 \msg_new:nnn { ducksay } { justify~unavailable }
327 {
328   Justified~content~is~not~available~for~tabular~argument~mode~without~fixed~
329   width.~'l'~column~is~used~instead.
330 }
331 \msg_new:nnn { ducksay } { unknown~message~alignment }
332 {
333   The~specified~message~alignment~'\exp_not:n { #1 }'~is~unknown.~
334   'l'~is~used~as~fallback.
335 }
336 \msg_new:nnn { ducksay } { v1~key~only }
337 { The~'\l_keys_key_tl'~key~is~only~available~for~'version=1'. }

```

### 2.3.2 Variables

#### 2.3.2.1 Token Lists

```

338 \tl_new:N \l_ducksay_msg_align_vbox_tl

```

#### 2.3.2.2 Boxes

```

339 \box_new:N \l_ducksay_msg_box

```

#### 2.3.2.3 Bools

```

340 \bool_new:N \l_ducksay_eat_arg_box_bool
341 \bool_new:N \l_ducksay_eat_arg_tab_verb_bool
342 \bool_new:N \l_ducksay_mirrored_body_bool

```

#### 2.3.2.4 Coffins

```

343 \coffin_new:N \l_ducksay_body_coffin
344 \coffin_new:N \l_ducksay_bubble_close_coffin
345 \coffin_new:N \l_ducksay_bubble_open_coffin
346 \coffin_new:N \l_ducksay_bubble_top_coffin
347 \coffin_new:N \l_ducksay_msg_coffin

```

#### 2.3.2.5 Dimensions

```

348 \dim_new:N \l_ducksay_hpad_dim
349 \dim_new:N \l_ducksay_bubble_bottom_kern_dim
350 \dim_new:N \l_ducksay_bubble_top_kern_dim
351 \dim_new:N \l_ducksay_msg_width_dim

```

### 2.3.3 Options

```

352 \keys_define:nn { ducksay }
353 {
354   ,arg .choice:
355   ,arg / box .code:n = \bool_set_true:N \l_ducksay_eat_arg_box_bool
356   ,arg / tab .code:n =
357   {

```

```

( 26 )
\ .-'\
" \
----- ( .)-----
)-----)-----)-----)-----)

```

```

358     \bool_set_false:N \l_ducksay_eat_arg_box_bool
359     \bool_set_false:N \l_ducksay_eat_arg_tab_verb_bool
360 }
361 ,arg / tab* .code:n =
362 {
363     \bool_set_false:N \l_ducksay_eat_arg_box_bool
364     \bool_set_true:N \l_ducksay_eat_arg_tab_verb_bool
365 }
366 ,arg .initial:n = tab
367 ,wd* .dim_set:N = \l_ducksay_msg_width_dim
368 ,wd* .initial:n = -\c_max_dim
369 ,wd* .value_required:n = true
370 ,none .bool_set:N = \l_ducksay_no_body_bool
371 ,no-bubble .bool_set:N = \l_ducksay_no_bubble_bool
372 ,body-mirrored .bool_set:N = \l_ducksay_mirrored_body_bool
373 ,ignore-body .bool_set:N = \l_ducksay_ignored_body_bool
374 ,body-x .dim_set:N = \l_ducksay_body_x_offset_dim
375 ,body-x .value_required:n = true
376 ,body-y .dim_set:N = \l_ducksay_body_y_offset_dim
377 ,body-y .value_required:n = true
378 ,body-to-msg .tl_set:N = \l_ducksay_body_to_msg_align_body_tl
379 ,msg-to-body .tl_set:N = \l_ducksay_body_to_msg_align_msg_tl
380 ,body-align .choice:
381 ,body-align / l .meta:n = { body-to-msg = l , msg-to-body = l }
382 ,body-align / c .meta:n = { body-to-msg = hc , msg-to-body = hc }
383 ,body-align / r .meta:n = { body-to-msg = r , msg-to-body = r }
384 ,body-align .initial:n = l
385 ,body-bigger .int_set:N = \l_ducksay_body_bigger_int
386 ,body-bigger .initial:n = \c_zero
387 ,msg-align .choice:
388 ,msg-align / l .code:n = { \tl_set:Nn \l_ducksay_msg_align_tl { l } }
389 ,msg-align / c .code:n = { \tl_set:Nn \l_ducksay_msg_align_tl { c } }
390 ,msg-align / r .code:n = { \tl_set:Nn \l_ducksay_msg_align_tl { r } }
391 ,msg-align / j .code:n = { \tl_set:Nn \l_ducksay_msg_align_tl { j } }
392 ,msg-align-l .tl_set:N = \l_ducksay_msg_align_l_tl
393 ,msg-align-l .initial:n = \raggedright
394 ,msg-align-c .tl_set:N = \l_ducksay_msg_align_c_tl
395 ,msg-align-c .initial:n = \centering
396 ,msg-align-r .tl_set:N = \l_ducksay_msg_align_r_tl
397 ,msg-align-r .initial:n = \raggedleft
398 ,msg-align-j .tl_set:N = \l_ducksay_msg_align_j_tl
399 ,msg-align-j .initial:n = {}
400 ,out-h .tl_set:N = \l_ducksay_output_h_pole_tl
401 ,out-h .initial:n = l
402 ,out-v .tl_set:N = \l_ducksay_output_v_pole_tl
403 ,out-v .initial:n = vc
404 ,out-x .dim_set:N = \l_ducksay_output_x_offset_dim
405 ,out-x .value_required:n = true
406 ,out-y .dim_set:N = \l_ducksay_output_y_offset_dim
407 ,out-y .value_required:n = true
408 ,t .meta:n = { out-v = t }
409 ,c .meta:n = { out-v = vc }
410 ,b .meta:n = { out-v = b }
411 ,body* .tl_set:N = \l_ducksay_body_fount_tl

```

```

412 ,msg* .tl_set:N = \l_ducksay_msg_fount_tl
413 ,bubble* .tl_set:N = \l_ducksay_bubble_fount_tl
414 ,body* .initial:n = \verbatim@font
415 ,msg* .initial:n = \verbatim@font
416 ,bubble* .initial:n = \verbatim@font
417 ,body .code:n = \tl_put_right:Nn \l_ducksay_body_fount_tl { #1 }
418 ,msg .code:n = \tl_put_right:Nn \l_ducksay_msg_fount_tl { #1 }
419 ,bubble .code:n = \tl_put_right:Nn \l_ducksay_bubble_fount_tl { #1 }
420 ,MSG .meta:n = { msg = #1 , bubble = #1 }
421 ,MSG* .meta:n = { msg* = #1 , bubble* = #1 }
422 ,hpad .int_set:N = \l_ducksay_hpad_int
423 ,hpad .initial:n = 2
424 ,hpad .value_required:n = true
425 ,vpad .int_set:N = \l_ducksay_vpad_int
426 ,vpad .value_required:n = true
427 ,col .tl_set:N = \l_ducksay_msg_tabular_column_tl
428 ,bubble-top-kern .tl_set:N = \l_ducksay_bubble_top_kern_tl
429 ,bubble-top-kern .initial:n = { -.5ex }
430 ,bubble-top-kern .value_required:n = true
431 ,bubble-bot-kern .tl_set:N = \l_ducksay_bubble_bottom_kern_tl
432 ,bubble-bot-kern .initial:n = { .2ex }
433 ,bubble-bot-kern .value_required:n = true
434 ,bubble-side-kern .tl_set:N = \l_ducksay_bubble_side_kern_tl
435 ,bubble-side-kern .initial:n = { .2em }
436 ,bubble-side-kern .value_required:n = true
437 ,bubble-delim-top .tl_set:N = \l_ducksay_bubble_delim_top_tl
438 ,bubble-delim-left-1 .tl_set:N = \l_ducksay_bubble_delim_left_a_tl
439 ,bubble-delim-left-2 .tl_set:N = \l_ducksay_bubble_delim_left_b_tl
440 ,bubble-delim-left-3 .tl_set:N = \l_ducksay_bubble_delim_left_c_tl
441 ,bubble-delim-left-4 .tl_set:N = \l_ducksay_bubble_delim_left_d_tl
442 ,bubble-delim-right-1 .tl_set:N = \l_ducksay_bubble_delim_right_a_tl
443 ,bubble-delim-right-2 .tl_set:N = \l_ducksay_bubble_delim_right_b_tl
444 ,bubble-delim-right-3 .tl_set:N = \l_ducksay_bubble_delim_right_c_tl
445 ,bubble-delim-right-4 .tl_set:N = \l_ducksay_bubble_delim_right_d_tl
446 ,bubble-delim-top .initial:n = { { - } }
447 ,bubble-delim-left-1 .initial:n = (
448 ,bubble-delim-left-2 .initial:n = /
449 ,bubble-delim-left-3 .initial:n = |
450 ,bubble-delim-left-4 .initial:n = \c_backslash_str
451 ,bubble-delim-right-1 .initial:n = )
452 ,bubble-delim-right-2 .initial:n = \c_backslash_str
453 ,bubble-delim-right-3 .initial:n = |
454 ,bubble-delim-right-4 .initial:n = /
455 ,strip-spaces .bool_set:N = \l_ducksay_msg_strip_spaces_bool
456 }

```

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

```

457 \clist_map_inline:nn
458 { align, rel-align }
459 {
460   \keys_define:nn { ducksay }
461     { #1 .code:n = \msg_error:nn { ducksay } { v1-key-only } }
462 }

```

### 2.3.4 Functions

#### 2.3.4.1 Internal

```
evaluate_message_alignment_fixed_width_common:
```

```

463 \cs_new:Npn \ducksay_evaluate_message_alignment_fixed_width_common:
464 {
465   \str_case:Vn \l_ducksay_msg_align_tl
466   {
467     { l } { \exp_not:N \l_ducksay_msg_align_l_tl }
468     { c } { \exp_not:N \l_ducksay_msg_align_c_tl }
469     { r } { \exp_not:N \l_ducksay_msg_align_r_tl }
470     { j } { \exp_not:N \l_ducksay_msg_align_j_tl }
471   }
472 }

```

(End definition for \ducksay\_evaluate\_message\_alignment\_fixed\_width\_common:. This function is documented on page ??.)

```
evaluate_message_alignment_fixed_width_tabular:
```

```

473 \cs_new:Npn \ducksay_evaluate_message_alignment_fixed_width_tabular:
474 {
475   \tl_if_empty:NT \l_ducksay_msg_tabular_column_tl
476   {
477     \tl_set:Nx \l_ducksay_msg_tabular_column_tl
478     {
479       >
480       {
481         \ducksay_evaluate_message_alignment_fixed_width_common:
482         \exp_not:N \arraybackslash
483       }
484       p { \exp_not:N \l_ducksay_msg_width_dim }
485     }
486   }
487 }

```

(End definition for \ducksay\_evaluate\_message\_alignment\_fixed\_width\_tabular:. This function is documented on page ??.)

```
evaluate_message_alignment_fixed_width_vbox:
```

```

488 \cs_new:Npn \ducksay_evaluate_message_alignment_fixed_width_vbox:
489 {
490   \tl_set:Nx \l_ducksay_msg_align_vbox_tl
491     { \ducksay_evaluate_message_alignment_fixed_width_common: }
492 }

```

(End definition for \ducksay\_evaluate\_message\_alignment\_fixed\_width\_vbox:. This function is documented on page ??.)

```
\ducksay calculate msg width from int:
```

```

493 \cs_new:Npn \ducksay_calculate_msg_width_from_int:
494 {
495   \hbox_set:Nn \l_ducksay_tmpa_box { { \l_ducksay_msg_fount_tl M } }
496   \dim_set:Nn \l_ducksay_msg_width_dim
497     { \l_ducksay_msg_width_int \box_wd:N \l_ducksay_tmpa_box }
498 }

```

(End definition for \ducksay\_calculate\_msg\_width\_from\_int:. This function is documented on page ??.)

\ducksay\_msg\_tabular\_begin:

```

499 \cs_new:Npn \ducksay_msg_tabular_begin:
500 {
501   \ducksay_msg_tabular_begin_inner:V \l_ducksay_msg_tabular_column_tl
502 }
503 \cs_new:Npn \ducksay_msg_tabular_begin_inner:n #1
504 {
505   \begin { tabular } { @{} #1 @{} }
506 }
507 \cs_generate_variant:Nn \ducksay_msg_tabular_begin_inner:n { V }

```

(End definition for \ducksay\_msg\_tabular\_begin:. This function is documented on page ??.)

\ducksay\_msg\_tabular\_end:

```

508 \cs_new:Npn \ducksay_msg_tabular_end:
509 {
510   \end { tabular }
511 }

```

(End definition for \ducksay\_msg\_tabular\_end:. This function is documented on page ??.)

\ducksay\_width\_case\_none\_int\_dim:nnn

```

512 \cs_new:Npn \ducksay_width_case_none_int_dim:nnn #1 #2 #3
513 {
514   \dim_compare:nNnTF { \l_ducksay_msg_width_dim } < { \c_zero_dim }
515   {
516     \int_compare:nNnTF { \l_ducksay_msg_width_int } < { \c_zero_int }
517     { #1 }
518     { #2 }
519   }
520   { #3 }
521 }

```

(End definition for \ducksay\_width\_case\_none\_int\_dim:nnn. This function is documented on page ??.)

\ducksay\_digest\_options:n

```

522 \cs_new:Npn \ducksay_digest_options:n #1
523 {
524   \group_begin:
525   \keys_set:nn { ducksay } { #1 }
526   \tl_if_empty:NT \l_ducksay_animal_tl
527   { \keys_set:nn { ducksay } { default_animal } }
528   \bool_if:NTF \l_ducksay_eat_arg_box_bool
529   {
530     \ducksay_width_case_none_int_dim:nnn
531     { \ducksay_eat_argument_hbox:w }
532     {
533       \ducksay_calculate_msg_width_from_int:
534       \ducksay_eat_argument_vbox:w
535     }
536     { \ducksay_eat_argument_vbox:w }
537   }

```

```

(-----)
\ .-' \
" \
----- ( '-----
)-----)

```

```

538 {
539   \ducksay_width_case_none_int_dim:nnn
540   {
541     \tl_if_empty:NT \l_ducksay_msg_tabular_column_tl
542     {
543       \str_case:Vn \l_ducksay_msg_align_tl
544       {
545         { l } { \tl_set:Nn \l_ducksay_msg_tabular_column_tl { l } }
546         { c } { \tl_set:Nn \l_ducksay_msg_tabular_column_tl { c } }
547         { r } { \tl_set:Nn \l_ducksay_msg_tabular_column_tl { r } }
548         { j }
549         {
550           \msg_error:nn { ducksay } { justify-unavailable }
551           \tl_set:Nn \l_ducksay_msg_tabular_column_tl { l }
552         }
553       }
554     }
555   }
556   {
557     \ducksay_calculate_msg_width_from_int:
558     \ducksay_evaluate_message_alignment_fixed_width_tabular:
559   }
560   { \ducksay_evaluate_message_alignment_fixed_width_tabular: }
561   \ducksay_eat_argument_tabular:w
562 }
563 }

```

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

\ducksay\_set\_bubble\_top\_kern:

```

564 \cs_new:Npn \ducksay_set_bubble_top_kern:
565 {
566   \group_begin:
567   \l_ducksay_bubble_fount_tl
568   \exp_args:NNNx
569   \group_end:
570   \dim_set:Nn \l_ducksay_bubble_top_kern_dim
571   { \dim_eval:n { \l_ducksay_bubble_top_kern_tl } }
572 }

```

(End definition for \ducksay\_set\_bubble\_top\_kern:. This function is documented on page ??.)

\ducksay\_set\_bubble\_bottom\_kern:

```

573 \cs_new:Npn \ducksay_set_bubble_bottom_kern:
574 {
575   \group_begin:
576   \l_ducksay_bubble_fount_tl
577   \exp_args:NNNx
578   \group_end:
579   \dim_set:Nn \l_ducksay_bubble_bottom_kern_dim
580   { \dim_eval:n { \l_ducksay_bubble_bottom_kern_tl } }
581 }

```

(End definition for \ducksay\_set\_bubble\_bottom\_kern:. This function is documented on page ??.)



\ducksay\_make\_body\_bigger:

```

582 \cs_new:Npn \ducksay_make_body_bigger:
583 {
584   \int_step_function:nN \l_ducksay_body_bigger_int
585   \ducksay_make_body_bigger_aux:n
586 }

```

(End definition for \ducksay\_make\_body\_bigger:. This function is documented on page ??.)

\ducksay\_make\_body\_bigger\_aux:n

```

587 \cs_new:Npn \ducksay_make_body_bigger_aux:n #1
588 {
589   \\\
590 }

```

(End definition for \ducksay\_make\_body\_bigger\_aux:n. This function is documented on page ??.)

\ducksay\_shipout:

```

591 \cs_new_protected:Npn \ducksay_shipout:
592 {
593   \hcoffin_set:Nn \l_ducksay_msg_coffin { \box_use:N \l_ducksay_msg_box }
594   \bool_if:NF \l_ducksay_no_bubble_bool
595   {
596     \hbox_set:Nn \l_ducksay_tmpa_box
597     { \l_ducksay_bubble_fount_tl \l_ducksay_bubble_delim_top_tl }
598     \int_set:Nn \l_ducksay_msg_width_int
599     {
600       \fp_eval:n
601       {
602         ceil
603         (
604           \box_wd:N \l_ducksay_msg_box / \box_wd:N \l_ducksay_tmpa_box
605         )
606       }
607     }
608     \group_begin:
609     \l_ducksay_bubble_fount_tl
610     \exp_args:NNNx
611     \group_end:
612     \int_set:Nn \l_ducksay_msg_height_int
613     {
614       \int_max:nn
615       {
616         \fp_eval:n
617         {
618           ceil
619           (
620             (
621               \box_ht:N \l_ducksay_msg_box
622               + \box_dp:N \l_ducksay_msg_box
623             )
624             / ( \arraystretch * \baselineskip )
625           )
626         }
627       }
628     }

```

```

627         + \l_ducksay_vpad_int
628     }
629     { \l_ducksay_msg_height_int }
630 }
631 \hcoffin_set:Nn \l_ducksay_bubble_open_coffin
632 {
633     \l_ducksay_bubble_fount_tl
634     \begin{tabular}{@{}l@{}}
635         \int_compare:nNnTF { \l_ducksay_msg_height_int } = { \c_one_int }
636         {
637             \l_ducksay_bubble_delim_left_a_tl
638         }
639         {
640             \l_ducksay_bubble_delim_left_b_tl\\
641             \int_step_inline:nnn
642                 { 3 } { \l_ducksay_msg_height_int }
643                 {
644                     \kern-\l_ducksay_bubble_side_kern_tl
645                     \l_ducksay_bubble_delim_left_c_tl
646                     \\
647                 }
648             \l_ducksay_bubble_delim_left_d_tl
649         }
650     \end{tabular}
651 }
652 \hcoffin_set:Nn \l_ducksay_bubble_close_coffin
653 {
654     \l_ducksay_bubble_fount_tl
655     \begin{tabular}{@{}r@{}}
656         \int_compare:nNnTF { \l_ducksay_msg_height_int } = { \c_one_int }
657         {
658             \l_ducksay_bubble_delim_right_a_tl
659         }
660         {
661             \l_ducksay_bubble_delim_right_b_tl \\
662             \int_step_inline:nnn
663                 { 3 } { \l_ducksay_msg_height_int }
664                 {
665                     \l_ducksay_bubble_delim_right_c_tl
666                     \kern-\l_ducksay_bubble_side_kern_tl
667                     \\
668                 }
669             \l_ducksay_bubble_delim_right_d_tl
670         }
671     \end{tabular}
672 }
673 \hcoffin_set:Nn \l_ducksay_bubble_top_coffin
674 {
675     \l_ducksay_bubble_fount_tl
676     \int_step_inline:nn
677         { \l_ducksay_msg_width_int + \l_ducksay_hpad_int }
678         { \l_ducksay_bubble_delim_top_tl }
679 }
680 \dim_set:Nn \l_ducksay_hpad_dim

```

```

(-----)
 \ .-' \
  " \
----- (-----)
)-----)

```

```

681     {
682     (
683         \coffin_wd:N \l_ducksay_bubble_top_coffin
684         - \coffin_wd:N \l_ducksay_msg_coffin
685     ) / 2
686     }
687 \coffin_join:NnnNnnnn
688 \l_ducksay_msg_coffin { l } { vc }
689 \l_ducksay_bubble_open_coffin { r } { vc }
690 { - \l_ducksay_hpad_dim } { \c_zero_dim }
691 \coffin_join:NnnNnnnn
692 \l_ducksay_msg_coffin { r } { vc }
693 \l_ducksay_bubble_close_coffin { l } { vc }
694 { \l_ducksay_hpad_dim } { \c_zero_dim }
695 \ducksay_set_bubble_top_kern:
696 \ducksay_set_bubble_bottom_kern:
697 \coffin_join:NnnNnnnn
698 \l_ducksay_msg_coffin { hc } { t }
699 \l_ducksay_bubble_top_coffin { hc } { b }
700 { \c_zero_dim } { \l_ducksay_bubble_top_kern_dim }
701 \coffin_join:NnnNnnnn
702 \l_ducksay_msg_coffin { hc } { b }
703 \l_ducksay_bubble_top_coffin { hc } { t }
704 { \c_zero_dim } { \l_ducksay_bubble_bottom_kern_dim }
705 }
706 \bool_if:NF \l_ducksay_no_body_bool
707 {
708     \hcoffin_set:Nn \l_ducksay_body_coffin
709     {
710         \frenchspacing
711         \l_ducksay_body_fount_tl
712         \begin{tabular} { @{} l @{} }
713             \l_ducksay_animal_tl
714             \ducksay_make_body_bigger:
715             \relax
716         \end{tabular}
717     }
718 \bool_if:NT \l_ducksay_mirrored_body_bool
719 {
720     \coffin_scale:Nnn \l_ducksay_body_coffin
721     { -\c_one_int } { \c_one_int }
722     \str_case:Vn \l_ducksay_body_to_msg_align_body_tl
723     {
724         { l } { \tl_set:Nn \l_ducksay_body_to_msg_align_body_tl { r } }
725         { r } { \tl_set:Nn \l_ducksay_body_to_msg_align_body_tl { l } }
726     }
727 }
728 \bool_if:NTF \l_ducksay_ignored_body_bool
729 { \coffin_attach:NVnNVnnn }
730 { \coffin_join:NVnNVnnn }
731 \l_ducksay_msg_coffin \l_ducksay_body_to_msg_align_msg_tl { b }
732 \l_ducksay_body_coffin \l_ducksay_body_to_msg_align_body_tl { t }
733 { \l_ducksay_body_x_offset_dim } { \l_ducksay_body_y_offset_dim }
734 }

```

```

735 \coffin_typeset:NVVnn \l_ducksay_msg_coffin
736 \l_ducksay_output_h_pole_tl \l_ducksay_output_v_pole_tl
737 { \l_ducksay_output_x_offset_dim } { \l_ducksay_output_y_offset_dim }
738 \group_end:
739 }

```

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

**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.

\ducksay\_eat\_argument\_tabular:w

```

740 \cs_new:Npn \ducksay_eat_argument_tabular:w
741 {
742   \bool_if:NTF \l_ducksay_eat_arg_tab_verb_bool
743   { \ducksay_eat_argument_tabular_verb:w }
744   { \ducksay_eat_argument_tabular_normal:w }
745 }

```

(End definition for \ducksay\_eat\_argument\_tabular:w. This function is documented on page ??.)

\ducksay\_eat\_argument\_tabular\_inner:w

```

746 \cs_new:Npn \ducksay_eat_argument_tabular_inner:w #1
747 {
748   \hbox_set:Nn \l_ducksay_msg_box
749   {
750     \l_ducksay_msg_fount_tl
751     \ducksay_msg_tabular_begin:
752     #1
753     \ducksay_msg_tabular_end:
754   }
755   \ducksay_shipout:
756 }

```

(End definition for \ducksay\_eat\_argument\_tabular\_inner:w. This function is documented on page ??.)

\ducksay\_eat\_argument\_tabular\_verb:w

```

757 \NewDocumentCommand \ducksay_eat_argument_tabular_verb:w
758 { >{ \ducksay_process_verb_newline:nnn { ~ } { ~ \par } } +v }
759 {
760   \ducksay_eat_argument_tabular_inner:w
761   {
762     \group_begin:
763     \__ducksay_everyeof:w { \exp_not:N }
764     \exp_after:wN
765     \group_end:
766     \__ducksay_scantokens:w { #1 }
767   }
768 }

```

(End definition for \ducksay\_eat\_argument\_tabular\_verb:w. This function is documented on page ??.)

```

( 35 )
\ .-' \
" \
----- ( .) -----
) -----) -----) -----)

```

`\ducksay_eat_argument_tabular_normal:w`

```
769 \NewDocumentCommand \ducksay_eat_argument_tabular_normal:w { +m }
770 { \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`

```
771 \cs_new_protected_nopar:Npn \ducksay_eat_argument_hbox:w
772 {
773   \bool_if:NTF \l_ducksay_msg_strip_spaces_bool
774   { \@grabbox }
775   { \@grabbox* }
776   {} \l_ducksay_msg_box \l_ducksay_msg_fount_tl \hbox {} \ducksay_shipout:
777 }
```

*(End definition for \ducksay\_eat\_argument\_hbox:w. This function is documented on page ??.)*

`\ducksay_eat_argument_vbox:w`

```
778 \cs_new_protected_nopar:Npn \ducksay_eat_argument_vbox:w
779 {
780   \ducksay_evaluate_message_alignment_fixed_width_vbox:
781   \bool_if:NTF \l_ducksay_msg_strip_spaces_bool
782   { \@grabbox }
783   { \@grabbox* }
784   {
785     \hsize \l_ducksay_msg_width_dim
786     \linewidth \hsize
787     \l_ducksay_msg_align_vbox_tl
788     \@afterindentfalse
789     \@afterheading
790   }
791   \l_ducksay_msg_box \l_ducksay_msg_fount_tl \vbox {} \ducksay_shipout:
792 }
```

*(End definition for \ducksay\_eat\_argument\_vbox:w. This function is documented on page ??.)*

### 2.3.4.1.2 Generating Variants of External Functions

```
793 \cs_generate_variant:Nn \coffin_join:NnnNnnnn { NVnNVnnn }
794 \cs_generate_variant:Nn \coffin_attach:NnnNnnnn { NVnNVnnn }
795 \cs_generate_variant:Nn \coffin_typeset:Nnnnn { NVVnn }
796 \cs_generate_variant:Nn \str_case:nn { Vn }
```

### 2.3.4.2 Document level

`\ducksay`

```
797 \NewDocumentCommand \ducksay { 0{ } }
798 {
799   \ducksay_digest_options:n { #1 }
800 }
```

*(End definition for \ducksay. This function is documented on page 8.)*

`\duckthink`

```
801 \NewDocumentCommand \duckthink { 0{} }
802 {
803   \ducksay_digest_options:n { think, #1 }
804 }
```

*(End definition for \duckthink. This function is documented on page 8.)*

```
805 \</code.v2>
```

## 2.4 Definition of the Animals

```

806 <*animals>
807 \ProvidesFile{ducksay.animals.tex}
808 [\ducksay@date\space v\ducksay@version\space ducksay animals]
809 %^~A some of the below are from http://ascii.co.uk/art/kangaroo
810 \AddAnimal{duck}%>>=
811 { \
812   \
813     >(' )
814     )/
815     /(
816     / '----/
817     \ ~-- /
818     ~~~~~~}%<<
819 \AddAnimal{small-duck}%>>=
820 { \
821   \
822     >()_
823     (__)__ _}%<<
824 \AddAnimal{duck-family}%>>=
825 { \
826   \
827     >(' )
828     )/
829     /(
830     / '----/ -()_ >()_
831     __\__~--/_/_ _()__ _}%<<
832 \AddAnimal{cow}%>>=
833 { \
834   \ ^__^
835     \ (oo)\_______
836         (__)\       )\/\
837         ||----w |
838         ||       ||}%<<
839 \AddAnimal{head-in}%>>=
840 { \
841   \ ^__^
842     \ (oo)\_______/ _____
843         (__)\       )= ( ____|_ \_____
844         ||----w | \ \ \ \ \_____
845         ||       ||       ||       ||}%<<
846 \AddAnimal{sodomized}%>>=
847 { \
848   \      ^
849     \ (oo)\_____/ \ \
850         (__)\       ) /
851         ||----w ((
852         ||       ||>>}%<<
853 \AddAnimal{tux}%>>=
854 { \
855   \ .--.
856     |o_o |
857     |\_/_ |

```

```

858     // \ \
859     (| | )
860     /'\_ _/'\
861     \___)=(___/}%=<<
862 \AddAnimal{pig}%>>=
863 + \_ _//| .-~-.
864 \_/_oo } }-@
865 ('')_ } |
866 '---| { }--{ }
867 //_/ /_/+}%=<<
868 \AddAnimal{frog}%>>=
869 { \
870 \ (.)_(.)
871 _ ( _ ) _
872 / \/'-----'\ \
873 --\ ( ( ) ) /--
874 ) \/\ _./ \/\ (
875 )_/ /\ \ /\ \ _({}%=<<
876 \AddAnimal{snowman}%>>=
877 { \
878 \_[_]_
879 (")
880 >-( : )-<
881 ( _ : _ )}%=<<
882 \AddAnimal[tail-symbol=s]{hedgehog}%>>=
883 { s .\|//| | | | .
884 s |/\| | | | | | | |
885 /. ' |/\| | | | | |
886 o _ , _ \| | | | | | | }'%=<<
887 \AddAnimal{kangaroo}%>>=
888 { \
889 \ _ , '
890 <--\ _ _ / \
891 \ _ / \ _ \
892 \ , \ / \ \
893 // \ \
894 ,/' ' \ _ ,}%=<<
895 %^~A http://chris.com/ascii/index.php?art=animals/rabbits
896 \AddAnimal[tail-symbol=s,tail-count=3]{rabbit}%>>=
897 { s
898 s / \ ' \
899 s | \ \ ' \ /' / \
900 \_/' \ \ -"-/' \ \
901 | | \ \ |
902 (d b) \_/
903 / \
904 ,".|.|. \_/.|.|. ",
905 / \/' _|_ ' \ \
906 | / ' _ ' " ' _ \ |
907 | | \ \ / / |
908 | \ \ \ / / |
909 \ \ \ / / /
910 ' " ' \ : / " '
911 ' " " " " " ' }'%=<<

```



```

912 \AddAnimal{bunny}%>=
913 { \
914     \      /
915     /\  /
916     ( )
917     .( o ).}%=<<
918 \AddAnimal{small-rabbit}%>=
919 { \
920     \ _//
921     (')---.
922     _/_( )o}%=<<
923 \AddAnimal[tail-symbol=s,tail-count=3]{dragon}%>=
924 {
925     s      | \_ _ / |      / \ \ // \
926     s      / 0  0  \_ _ /      // | \ \
927     /      / \ \_ /      // | \ \
928     @_~@' / \ \_ //      | \ \
929     // _ _ / \ \_ //      | \ \
930     ( // ) |      \ ///      | \ \
931     ( / / ) _ | _ / ) //      | \ \
932     ( // / ) ' / , _ _ / ( ; - .      | \ \
933     (( // / )) , - {      | . . . .      | \ \
934     (( // / )) ' \      /      | . . . .      | \ \
935     (( /// ))      ' . {      | . . . .      | \ \
936     (( / ))      . . . . . \      | . . . .      | \ \
937     /// . . . . . >      | . . . .      | \ \
938     /// - . . . . . } ~ - - - - -      | . . . .      | \ \
939                                     / . . . .      | \ \
940                                     / . . . .      | \ \
941                                     / . . . .      | \ \
942                                     / . . . .      | \ \
943                                     / . . . .      | \ \
944                                     / . . . .      | \ \
945                                     / . . . .      | \ \
946                                     / . . . .      | \ \
947                                     / . . . .      | \ \
948                                     / . . . .      | \ \
949                                     / . . . .      | \ \
950                                     / . . . .      | \ \
951                                     / . . . .      | \ \
952                                     / . . . .      | \ \
953                                     / . . . .      | \ \
954                                     / . . . .      | \ \
955                                     / . . . .      | \ \
956                                     / . . . .      | \ \
957                                     / . . . .      | \ \
958                                     / . . . .      | \ \
959                                     / . . . .      | \ \
960                                     / . . . .      | \ \
961                                     / . . . .      | \ \
962                                     / . . . .      | \ \
963                                     / . . . .      | \ \
964                                     / . . . .      | \ \
965                                     / . . . .      | \ \

```





```

1074      |      ||      |
1075      |-----||-----|
1076      | /  ----- \ /  ----- \ |
1077      /  (      )  (      )  \
1078      / \  ----- ()  ----- / \
1079      / \      /||\
1080      / \      /||||\
1081      / \      /|||||\
1082      /_      \0=====0/      \_
1083      '---..._|'-'-'-'|---...--'
1084      |      '      |}%=<<
1085      \AddAnimal[tail-symbol=|,tail-count=1]{crusader}%>>=
1086      { |
1087      \[T]/}
1088      \csname bool_if:cT\endcsname {l_ducksay_version_one_bool}
1089      {\AnimalOptions{crusader}{tail-1=|,rel-align=c}}
1090      \csname bool_if:cT\endcsname {l_ducksay_version_two_bool}
1091      {\AnimalOptions{crusader}{tail-1=|,body-align=c}}}%=<<
1092      %^A http://ascii.co.uk/art/knights
1093      \AddAnimal[tail-count=3]{knight}%>>=
1094      {
1095      \      ,-"-".
1096      \      |===|
1097      )      | (
1098      .==\' " "/\'==.
1099      .' \      (':') /'.
1100      _/_ |_.-' : '-._|__\
1101      <--->\'      : / \'<--->
1102      / /      >=====< / /
1103      _/_ .' / ,--:. \/=,'
1104      /_/_ |__v^v^v^v\__ \
1105      \(\) |V^V^V^V^V|\_/
1106      (\ \      \'---|---'/
1107      \ \      \-._|_,-/
1108      \ \      |__|__|
1109      \ \      <__X__>
1110      \ \      \..|.. /
1111      \ \      \ | /
1112      \ \      /V|V\
1113      \ /      | \
1114      '---' '---' }%=<<
1115      %^A https://www.asciart.eu/mythology/ghosts
1116      \AddAnimal{ghost}%>>=
1117      {
1118      \      .-.
1119      (o o)
1120      | 0 \
1121      \    \
1122      '~~~'}%=<<
1123      %^Ahttps://www.asciart.eu/index.php?art=creatures/fairies
1124      \AddAnimal{fairy}%>>=
1125      {
1126      \      .o00b
1127      ..      .o0      0

```

```

1128      ' : ; ; d      0
1129      ; ; ; ; d      . . o 0
1130      *      : : 0 ; ; ; ' 0 o o 0
1131      ~ " \ .  d p ' ( 0 . o .
1132      \ o p      ' o 0 b
1133      o b U
1134      d o p
1135      d o p
1136      P 0
1137      0 ' b
1138      1 P .
1139      / ;
1140      ' } % = < <
1141      \AddAnimal[tail-symbol=s]{only-tail}%>>=
1142      { s
1143      s } % = < <
1144      \AddAnimal[tail-symbol=s,tail-count=3]{only-tail3}%>>=
1145      { s
1146      s
1147      s } % = < <
1148      </animals>

```

Who's gonna use it anyway?

0

o

>( ' )

) /

/ (

/ ' ----- /

\ ~ = - /

~ ^ ~ ^ ~ ^ ~ ^ ~ ^ ~ ^ ~ ^

Hosted at  
[https://github.com/Skillmon/ltx\\_ducksay](https://github.com/Skillmon/ltx_ducksay)  
it is.

--.-.-  
'-.-"7'  
/'-c  
| /T  
\_)\_ /LI