Term-Image

Release 0.6.1

Toluwaleke Ogundipe

Apr 30, 2023

CONTENTS

1 Contents **3**

1.1 Getting Started . . . 3
1.2 User Guide . . . 14
1.3 API Reference . . . 20
1.4 Planned Features . . . 54
1.5 Known Issues . . . 55
1.6 FAQs . . . 55
1.7 Glossary . . . 56

2 Indices and Tables **61**

Python Module Index **63**

Index **65**

Attention: Under Construction - There might be incompatible changes between minor versions of version zero!

If you want to use this library in a project while it's still on version zero, ensure you pin the dependency to a specific minor version e.g `>=0.4,<0.5`.

On this note, you probably also want to switch to the specific documentation for the version you're using (somewhere at the lower left corner of this page).

CHAPTER

ONE

CONTENTS

1.1 Getting Started

1.1.1 Installation

Requirements

- Operating System: Unix / Linux / MacOS X / Windows (limited support, see the *FAQs*)
- Python >= 3.7
- A terminal emulator with **any** of the following:
  - support for the Kitty graphics protocol.
  - support for the iTerm2 inline image protocol.
  - full Unicode support and ANSI 24-bit color support

  Plans to support a wider variety of terminal emulators are in motion (see *Planned Features*).

Steps

The latest **stable** version can be installed from PyPI with:

```
pip install term-image
```

The **development** version can be installed with:

```
pip install git+https://github.com/AnonymouX47/term-image.git
```

Supported Terminal Emulators

Some terminals emulators that have been tested to meet the requirements for at least one render style include:

- **libvte**-based terminal emulators such as:
  - Gnome Terminal
  - Terminator
  - Tilix
- Kitty

- Konsole
- iTerm2
- WezTerm
- Alacritty
- Windows Terminal
- MinTTY (on Windows)
- Termux (on Android)

Note: If you've tested `term-image` on any other terminal emulator that meets all requirements, please mention the name in a new thread under this discussion.

Also, if you're having an issue with terminal support, you may report or check information about it in the discussion linked above.

Note: Some terminal emulators support 24-bit color escape sequences but have a 256-color pallete. This will limit color reproduction.

1.1.2 Tutorial

This is a basic introduction to using the library. Please refer to the *API Reference* for detailed description of the features and functionality provided by the library.

For this tutorial we'll be using the image below:

The image has a resolution of **288x288 pixels**.

Note: All the samples in this tutorial occured in a terminal window of **255 columns by 70 lines**.

Creating an Instance

Image instances can be created using the convinience functions *`AutoImage()`*, *`from_file()`* and *`from_url()`*, which automatically detect the best style supported by the terminal emulator.

Instances can also be created using the *Image Classes* directly via their respective constructors or *`from_file()`* and *`from_url()`* methods.

1. Initialize with a file path:

```
from term_image.image import from_file

image = from_file("path/to/python.png")
```

2. Initialize with a URL:

```
from term_image.image import from_url

image = from_url("https://raw.githubusercontent.com/AnonymouX47/term-image/main/
↪docs/source/resources/tutorial/python.png")
```

3. Initialize with a PIL (Pillow) image instance:

```
from PIL import Image
from term_image.image import AutoImage

img = Image.open("path/to/python.png")
image = AutoImage(img)
```

Rendering an Image

Rendering an image is the process of converting it (per-frame for *animated* images) into text (a string) which reproduces a representation or approximation of the image when written to the terminal.

Hint: To display the rendered image in the following steps, pass the string as an argument to `print()`.

There are two ways to render an image:

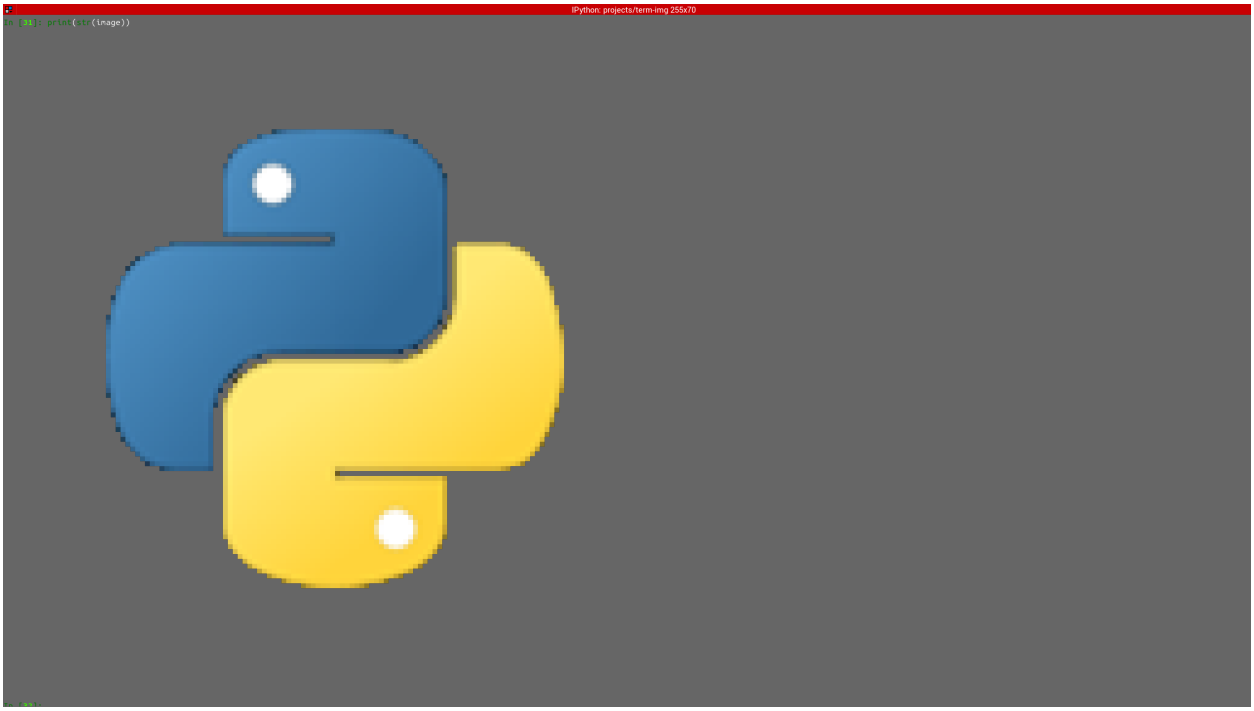
Unformatted Rendering

This is done using:

```
str(image)
```

The image is rendered without *padding*/*alignment* and with transparency enabled.

The output (using `print()`) should look like:



Formatted Rendering

Note: To see the effect of *alignment* in the steps below, please scale the image down using:

```
image.scale = 0.75
```

This simply sets the x-axis and y-axis *scale* of the image to `0.75`. We'll see more about this *later*.

Below are examples of formatted rendering:

```
format(image, "|200.^70#ffffff")
```

Renders the image with:

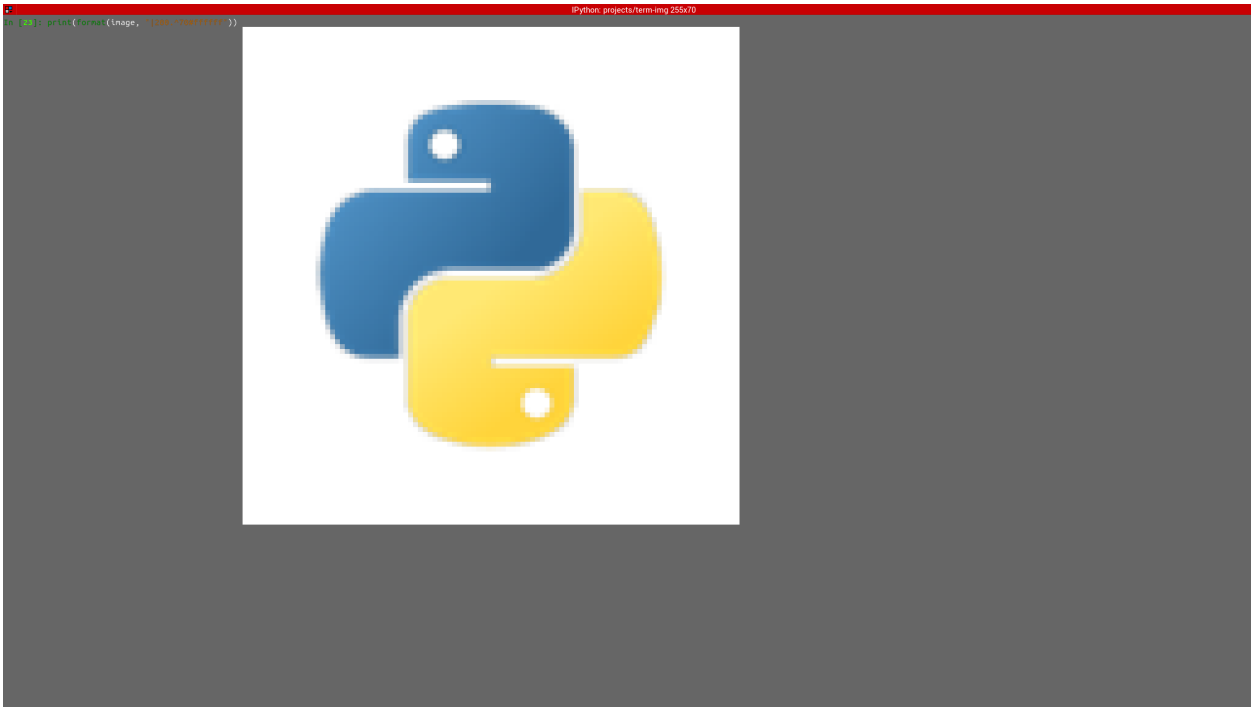
- **center** *horizontal alignment*
- a *padding width* of **200** columns

- **top** *vertical alignment*
- a *padding height* of **70** lines
- **white** (`#ffffff`) background underlay

Note: You might have to reduce the padding width (200) and/or height (70) to something that'll fit into your terminal window, or increase the size of the terminlal window

---

The output (using `print()`) should look like:

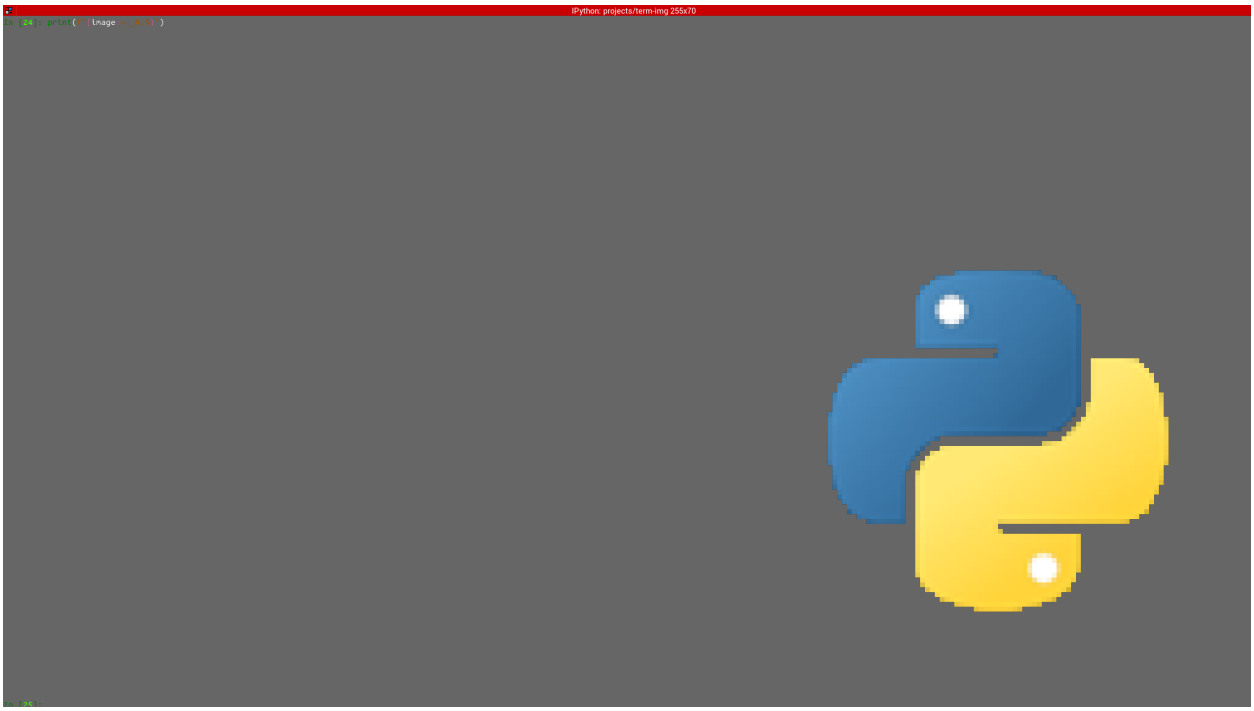


```
f"{image:>._#.5}"
```

Renders the image with:

- **right** *horizontal alignment*
- **automatic** *padding width* (the current *terminal width* minus *horizontal allowance*)
- **bottom** *vertical alignment*
- **automatic** *padding height* (the current *terminal height* minus *vertical allowance*)
- transparent background with **0.5** *alpha threshold*

The output (using `print()`) should look like:



```
"{:1.1#}".format(image)
```

Renders the image with:

- **center** *horizontal alignment* (default)
- **no** horizontal *padding*, since 1 is less than or equal to the image width
- **middle** *vertical alignment* (default)
- **no** vertical *padding*, since 1 is less than or equal to the image height
- transparency is **disabled** (alpha channel is ignored)

The output (using `print()`) should look like:

See also:

Render Formatting and *Render Format Specification*

Drawing/Displaying an Image

There are two basic ways to draw an image to the terminal screen:

1. Using the `draw()` method:

   ```
   image.draw()
   ```

   NOTE: `draw()` has various parameters for *Render Formatting*.

2. Using `print()` with an image render output (i.e printing the rendered string):

   ```
   print(image)  # Uses str()
   # OR
   print(f"{image:>200.^70#ffffff}")  # Uses format()
   ```

Note:

- For *animated* images, only the former animates the output, the latter only draws the **current** frame (see `seek()` and `tell()`).
- Also, the former performs size validation to see if the image will fit into the terminal, while the latter doesn't.

Important: All the examples above use *dynamic*, *automatic* sizing and default *scale*.

Image Size

The size of an image is the **unscaled** dimension with which an image is rendered.
The image size can be retrieved via the *`size`*, *`width`* and *`height`* properties.

The size of an image can be in either of two states:

1. Fixed

   In this state,

   - the `size` property evaluates to a 2-tuple of integers, while the `width` and `height` properties evaluate to integers,
   - the image is rendered with the set size.

2. Dynamic

   In this state,

   - the `size`, `width` and `height` properties evaluate to a *`Size`* enum member,
   - the size with which the image is rendered is automatically calculated (based on the current *terminal size* or the image's original size) whenever the image is to be rendered.

The size of an image can be set at instantiation by passing an integer or a *`Size`* enum member to **either** the *width* **or** the *height* **keyword-only** parameter. For whichever axis a dimension is given, the dimension on the other axis is calculated **proportionally**.

Note:

1. The arguments can only be given **by keyword**.
2. If neither is given, the *`FIT` dynamic size* applies.
3. All methods of instantiation accept these arguments.

For example:

```
>>> from term_image.image import Size, from_file
>>> image = from_file("python.png")  # Dynamic FIT
>>> image.size is Size.FIT
True
>>> image = from_file("python.png", width=60)  # Fixed
>>> image.size
(60, 30)
>>> image.height
30
>>> image = from_file("python.png", height=56)  # Fixed
>>> image.size
(112, 56)
>>> image.width
112
>>> image = from_file("python.png", height=Size.FIT)  # Fixed FIT
>>> image.size
(136, 68)
>>> image = from_file("python.png", width=Size.FIT_TO_WIDTH)  # Fixed FIT_TO_WIDTH
```

(continues on next page)

(continued from previous page)

```
>>> image.size
(255, 128)
>>> image = from_file("python.png", height=Size.ORIGINAL)  # Fixed ORIGINAL
>>> image.size
(288, 144)
```

No size validation is performed i.e the resulting size might not fit into the terminal window

```
>>> image = from_file("python.png", height=68)  # Will fit in, OK
>>> image.size
(136, 68)
>>> image = from_file("python.png", height=500)  # Will not fit in, also OK
>>> image.size
(1000, 500)
```

An exception is raised when both *width* and *height* are given.

```
>>> image = from_file("python.png", width=100, height=100)
Traceback (most recent call last):
  .
  .
  .
ValueError: Cannot specify both width and height
```

The *width* and *height* properties can be used to set the size of an image after instantiation, resulting in *fixed size*.

```
>>> image = from_file("python.png")
>>> image.width = 56
>>> image.size
(56, 28)
>>> image.height
28
>>> image.height = 68
>>> image.size
(136, 68)
>>> image.width
136
>>> # Even though the terminal can't contain the resulting height, the size is still set
>>> image.width = 200
>>> image.size
(200, 100)
>>> image.width = Size.FIT
>>> image.size
(136, 69)
>>> image.height = Size.FIT_TO_WIDTH
>>> image.size
(255, 128)
>>> image.height = Size.ORIGINAL
>>> image.size
(288, 144)
```

The *size* property can only be set to a *Size* enum member, resulting in *dynamic size*.

```
>>> image = from_file("python.png")
>>> image.size = Size.FIT
>>> image.size is image.width is image.height is Size.FIT
True
>>> image.size = Size.FIT_TO_WIDTH
>>> image.size is image.width is image.height is Size.FIT_TO_WIDTH
True
>>> image.size = Size.ORIGINAL
>>> image.size is image.width is image.height is Size.ORIGINAL
True
```

Important:

1. The currently set *cell ratio* is also taken into consideration when calculating sizes for images of *Text-based Render Styles*.
2. There is a **default** 2-line *vertical allowance*, to allow for shell prompts or the likes.

Tip: See *set_size()* for extended sizing control.

Image scale

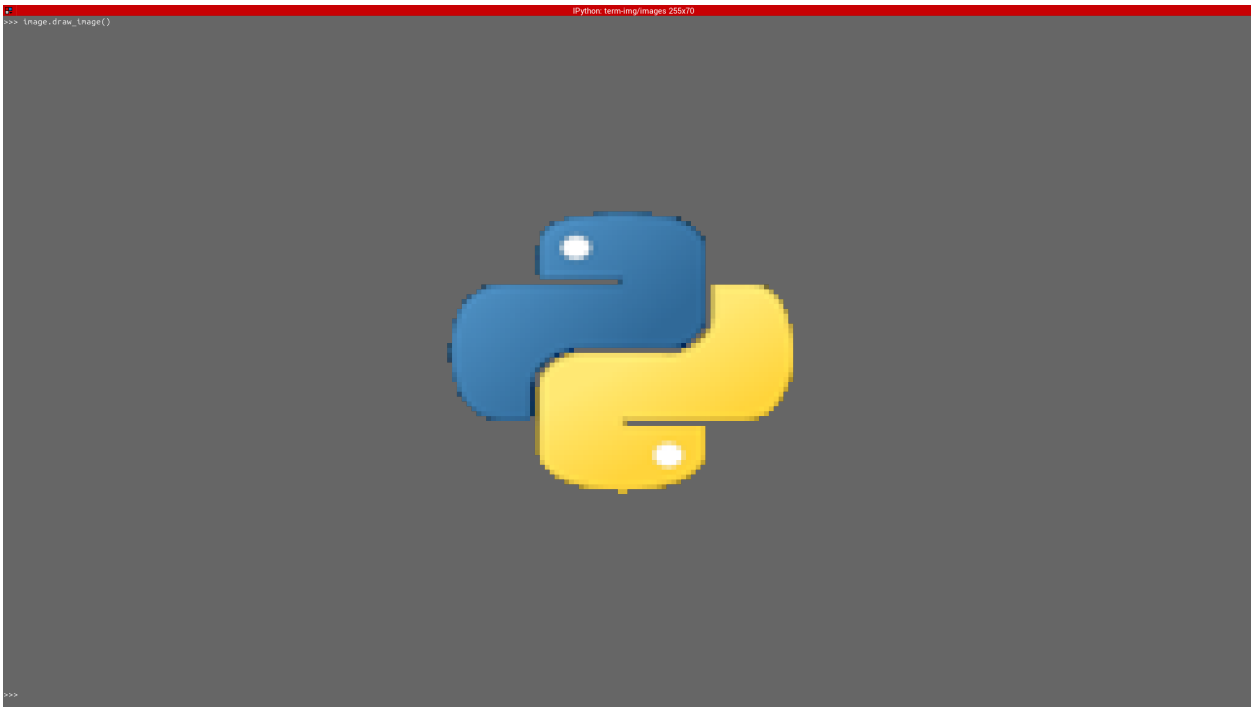
The scale of an image is the **ratio** of its size with which it will actually be rendered.
A valid scale value is a `float` in the range `0.0 < x <= 1.0` i.e greater than zero and less than or equal to one.

The image scale can be retrieved via the properties *scale*, *scale_x* and *scale_y*.

The scale can be set at instantiation by passing a value to the *scale* **keyword-only** paramter.

```
>>> image = from_file("python.png", scale=(0.75, 0.6))
>>> image.scale
>>> (0.75, 0.6)
```

The drawn image (using `image.draw()`) should look like:



If the *scale* argument is ommited, the default scale `(1.0, 1.0)` is used.

```
>>> image = from_file("python.png")
>>> image.scale
>>> (1.0, 1.0)
```

The drawn image (using `image.draw()`) should look like:

The properties *scale*, *scale_x* and *scale_y* are used to set the scale of an image after instantiation.

`scale` accepts a tuple of two scale values or a single scale value.
`scale_x` and `scale_y` each accept a single scale value.

```
>>> image = from_file("python.png")
>>> image.scale = (.3, .56756)
>>> image.scale
(0.3, 0.56756)
>>> image.scale = .5
>>> image.scale
(0.5, 0.5)
>>> image.scale_x = .75
>>> image.scale
(0.75, 0.5)
>>> image.scale_y = 1.
>>> image.scale
(0.75, 1.0)
```

Finally, to explore more of the library's features and functionality, check out the *User Guide* and the *API Reference*.

1.2 User Guide

1.2.1 Concepts

Render Styles

See *render style*.

All render style classes are designed to share a common interface (with some having extensions), making the usage of one class directly compatibile with another, except when using style-specific features.

Hence, the covenience functions *AutoImage*, *from_file* and *from_url* provide a means of render-style-agnostic usage of the library. These functions automatically detect the best render style supported by the *active terminal*.

There are two main categories of render styles:

Text-based Render Styles

Represent images using ASCII or Unicode symbols, and in some cases, with escape sequences to reproduce color.

Render style classes in this category are subclasses of *`TextImage`*. These include:

- *`BlockImage`*

Graphics-based Render Styles

Represent images with actual pixels, using terminal graphics protocols.

Render style classes in this category are subclasses of *`GraphicsImage`*. These include:

- *`KittyImage`*
- *`ITerm2Image`*

Render Methods

A *render style* may implement multiple *render methods*. See the **Render Methods** section in the description of a render style class (that implements multiple render methods), for the description of its render methods.

Auto Cell Ratio

Note: This concerns *Text-based Render Styles* only.

The is a feature which when supported, can be used to determine the *cell ratio* directly from the terminal emulator itself. With this feature, it is possible to always produce images of text-based render styles with correct **aspect ratio**.

When using either mode of *`AutoCellRatio`*, it's important to note that some terminal emulators (most non-graphics-capable ones) might have queried. See *Terminal Queries*.

If the program will never expect any useful input, particularly **while an image's size is being set/calculated**, then using *`DYNAMIC`* mode is OK. For an image with *dynamic size*, this includes when it's being rendered and when its *`rendered_size`*, *`rendered_width`* or *`rendered_height`* property is invoked.

Otherwise i.e if the program will be expecting input, use *`FIXED`* mode and use *`read_tty_all()`* to read all currently unread input just before calling *`set_cell_ratio()`*.

The Active Terminal

See *active terminal*.

The following streams/files are checked in the following order (along with the rationale behind the ordering):

- `STDOUT`: Since it's where images will most likely be drawn.
- `STDIN`: If output is redirected to a file or pipe and the input is a terminal, then using it as the *active terminal* should give the expected result i.e the same as when output is not redirected.
- `STDERR`: If both output and input are redirected, it's usually unlikely for errors to be.
- `/dev/tty`: Finally, if all else fail, fall back to the process' controlling terminal, if any.

The first one that is ascertained to be a terminal device is used for all *Terminal Queries* and to retrieve the terminal (and window) size on some terminal emulators.

Note: If none of the streams/files is a TTY device, then a *TermImageWarning* is issued and dependent functionality is disabled.

Terminal Queries

Some features of this library require the aquisition of certain information from the *active terminal*. A single iteration of this aquisition procedure is called a **query**.

A query involves three major steps:

1. Clear all unread input from the terminal
2. Write to the terminal
3. Read from the terminal

For this procedure to be successful, it must not be interrupted.

About #1
: If the program is expecting input, use *read_tty_all()* to read all currently unread input (**without blocking**) just before any operation involving a query.

About #2 and #3
: After sending a request to the terminal, its response is awaited. The default wait time is *DEFAULT_QUERY_TIMEOUT* but can be changed using *set_query_timeout()*. If the terminal emulator responds after the set timeout, this can result in the application program recieving what would seem to be garbage or ghost input (see this *FAQ*).

  If the program includes any other function that could write to the terminal OR especially, read from the terminal or modify it's attributes, while a query is in progress (as a result of asynchronous execution e.g multithreading or multiprocessing), decorate it with *lock_tty()* to ensure it doesn't interfere.

  For example, an image viewer based on this project uses urwid which reads from the terminal using `urwid.raw_display.Screen.get_available_raw_input()`. To prevent this method from interfering with terminal queries, it uses *UrwidImageScreen* which overrides and wraps the method like:

```
class UrwidImageScreen(Screen):
    @lock_tty
    def get_available_raw_input(self):
       return super().get_available_raw_input()
```

  Also, if the *active terminal* is not the controlling terminal of the process using this library (e.g output is redirected to another TTY device), ensure no process that can interfere with a query (e.g a shell or REPL) is currently running in the active terminal. For instance, such a process can be temporarily put to sleep.

Features that require terminal queries

In parentheses are the outcomes when the terminal doesn't support queries or when queries are disabled.

- *Auto Cell Ratio* (determined to be unsupported)
- Support checks for *Graphics-based Render Styles* (determined to be unsupported)
- Auto background color (black is used)
- Alpha blend for pixels above the alpha threshold in transparent renders with *Text-based Render Styles* (black is used)
- Workaround for ANSI background colors in text-based renders on the Kitty terminal (the workaround is disabled)

Note: This list might not always be complete. In case you notice

- any difference with any unlisted feature when terminal queries are enabled versus when disabled, or
- a behaviour different from the one specified for the listed features, when terminal queries are disabled,

please open an issue here.

1.2.2 Render Formatting

Render formatting is simply the modification of a primary *render* output. This is provided via:

- Python's string formatting protocol by using `format()`, `str.format()` or formatted string literals with the *Render Format Specification*
- Parameters of `draw()`

The following constitute render formatting:

Padding

This adds whitespace around a primary *render* output. The amount of whitespace added is determined by two values (with respect to the rendered size):

- *padding width*, determines horizontal padding
  - uses the `width` field of the *Render Format Specification*
  - uses the *pad_width* parameter of `draw()`
- *padding height*, determines vertical padding
  - uses the `height` field of the *Render Format Specification*
  - uses the *pad_height* parameter of `draw()`

If the padding width or height is less than or equal to the width or height of the primary render output, then the padding has no effect on the corresponding axis.

Alignment

This determines the position of a primary *render* output within it's *Padding*. The position is determined by two values:

- *horizontal alignment*, determines the horizontal position
  - uses the `h_align` field of the *Render Format Specification*
  - uses the *h_align* parameter of `draw()`
- *vertical alignment*, determines the vertical position
  - uses the `v_align` field of the *Render Format Specification*
  - uses the *v_align* parameter of `draw()`

Transparency

This determines how transparent pixels are rendered. Transparent pixels can be rendered in one of the following ways:

- Transparency disabled

  Alpha channel is ignored.
  - uses the `#` field of the *Render Format Specification*, without `threshold` or `bgcolor`
  - uses the *alpha* parameter of `draw()`, set to `None`
- Transparency enabled with an *alpha threshold*

  For *Text-based Render Styles*, any pixel with an alpha value above the given threshold is taken as **opaque**. For *Graphics-based Render Styles*, the alpha value of each pixel is used as-is.
  - uses the `threshold` field of the *Render Format Specification*
  - uses the *alpha* parameter of `draw()`, set to a `float` value
- Transparent pixels overlaid on a color

  May be specified to be a specific color or the default background color of the terminal emulator (if it can't be determined, black is used).
  - uses the `bgcolor` field of the *Render Format Specification*
  - uses the *alpha* parameter of `draw()`, set to a string value

Render Format Specification

```
[ <h_align> ]  [ <width> ]  [ . [ <v_align> ] [ <height> ] ]  [ # [ <threshold> |
↪<bgcolor> ] ]  [ + <style> ]
```

Note:

- spaces are only for clarity and not included in the syntax
- `<...>` is a placeholder for a single field
- `|` implies mutual exclusivity
- fields within `[ ]` are optional
- fields within `{ }` are required, though subject to any enclosing `[ ]`

- if the `.` is present, then at least one of `v_align` and `height` must be present

---

- `h_align` → *horizontal alignment*
  - `<` → left
  - `|` → center
  - `>` → right
  - *default* → center
- `width` → *padding width*
  - positive integer
  - *default*: *terminal width* minus *horizontal allowance*
  - if **less than or equal** to the *rendered width*, it has **no effect**
- `v_align` → *vertical alignment*
  - `^` → top
  - `-` → middle
  - `_` → bottom
  - *default* → middle
- `height` → *padding height*
  - positive integer
  - *default*: *terminal height* minus *vertical allowance*
  - if **less than or equal** to the *rendered height*, it has **no effect**
- `#` → transparency setting
  - *default*: transparency is enabled with the default *alpha threshold*
  - `threshold` → *alpha threshold*
    - a float value in the range `0.0` <= `threshold` < `1.0` (but starting with the `.` (decimal point))
    - **applies to only** *Text-based Render Styles*
    - e.g `.0`, `.325043`, `.999`
  - `bgcolor` → background underlay color
    - `#` → the terminal emulator's default background color (or black, if undetermined), OR
    - a hex color e.g `ffffff`, `7faa52`
  - if neither `threshold` nor `bgcolor` is present, but `#` is present, transparency is disabled i.e alpha channel is ignored
- `style` → style-specific format specifier

  See each render style in *Image Classes* for its own specification, if it defines.

  `style` can be broken down into `[ <parent> ] [ <current> ]`, where `current` is the spec defined by a render style and `parent` is the spec defined by a parent of that render style. `parent` can in turn be **recursively** broken down as such.

See also:

Formatted rendering tutorial.

1.3 API Reference

Attention: Under Construction - There might be incompatible interface changes between minor versions of version zero!

If you want to use the library in a project while it's still on version zero, ensure you pin the dependency to a specific minor version e.g `>=0.4,<0.5`.

On this note, you probably also want to switch to the specific documentation for the version you're using (somewhere at the lower left corner of this page).

Attention: Any module or definition not documented here should be considered part of the private interface and can be changed or removed at any time without notice.

1.3.1 Top-Level Definitions

Constants

term_image.**DEFAULT_QUERY_TIMEOUT**: float = 0.1

Default timeout for *Terminal Queries*

See also: *set_query_timeout()*

Enumerations

class term_image.**AutoCellRatio**(*value*)

Bases: enum.Enum

Values for setting *Auto Cell Ratio*.

is_supported: bool | None = None

Auto cell ratio support status. Can be

- `None` -> support status not yet determined
- `True` -> supported
- `False` -> not supported

Can be explicitly set when using auto cell ratio but want to avoid the support check in a situation where the support status is foreknown. Can help to avoid being wrongly detected as unsupported on a *queried* terminal that doesn't respond on time.

For instance, when using multiprocessing, if the support status has been determined in the main process, this value can simply be passed on to and set within the child processes.

FIXED

DYNAMIC

See *set_cell_ratio()*.

Functions

disable_queries	Disables *Terminal Queries*.
disable_win_size_swap	Disables a workaround for terminal emulators that wrongly report window dimensions swapped.
enable_queries	Re-Enables *Terminal Queries*.
enable_win_size_swap	Enables a workaround for terminal emulators that wrongly report window dimensions swapped.
get_cell_ratio	Returns the global *cell ratio*.
set_cell_ratio	Sets the global *cell ratio*.
set_query_timeout	Sets the timeout for *Terminal Queries*.

term_image.**disable_queries**()

Disables *Terminal Queries*.

To re-enable queries, call *enable_queries()*.

Note: This affects all *dependent features*.

term_image.**disable_win_size_swap**()

Disables a workaround for terminal emulators that wrongly report window dimensions swapped.

This workaround is disabled by default. While disabled, the window dimensions reported by the *active terminal* are used as-is.

Note: This affects *Auto Cell Ratio* computation and size computations for *Graphics-based Render Styles*.

term_image.**enable_queries**()

Re-Enables *Terminal Queries*.

Queries are enabled by default. To disable, call *disable_queries()*.

Note: This affects all *dependent features*.

term_image.**enable_win_size_swap**()

Enables a workaround for terminal emulators that wrongly report window dimensions swapped.

While enabled, the window dimensions reported by the *active terminal* are swapped. This workaround is required on some older VTE-based terminal emulators.

Note: This affects *Auto Cell Ratio* computation and size computations for *Graphics-based Render Styles*.

term_image.**get_cell_ratio**()

Returns the global *cell ratio*.

See *set_cell_ratio()*.

term_image.**set_cell_ratio**(*ratio*)

Sets the global *cell ratio*.

Parameters

ratio (*float* | term_image.AutoCellRatio) – Can be one of the following values.

- A positive float value.
- *AutoCellRatio.FIXED*, the ratio is immediately determined from the *active terminal*.
- *AutoCellRatio.DYNAMIC*, the ratio is determined from the *active terminal* whenever *get_cell_ratio()* is called, though with some caching involved, such that the ratio is re-determined only if the terminal size changes.

Raises

- **TypeError** – An argument is of an inappropriate type.
- **ValueError** – An argument is of an appropriate type but has an unexpected/invalid value.
- ***term_image.exceptions.TermImageError*** – Auto cell ratio is not supported in the *active terminal* or on the current platform.

This value is taken into consideration when setting image sizes for **text-based** render styles, in order to preserve the aspect ratio of images drawn to the terminal.

Note: Changing the cell ratio does not automatically affect any image that has a *fixed size*. For a change in cell ratio to take effect, the image's size has to be re-set.

Attention: See *Auto Cell Ratio* for details about the auto modes.

term_image.**set_query_timeout**(*timeout*)

Sets the timeout for *Terminal Queries*.

Parameters

timeout (*float*) – Time limit for awaiting a response from the terminal, in seconds.

Raises

- **TypeError** – *timeout* is not a float.
- **ValueError** – *timeout* is less than or equal to zero.

1.3.2 `image` Module

Functions

These functions automatically detect the best supported render style for the current terminal.

Since all classes share a common interface (as defined by *BaseImage*), any operation supported by one image class can be performed on any other image class, except style-specific operations.

auto_image_class	Selects the image render style that best suits the current terminal emulator.
AutoImage	Creates an image instance from a PIL image instance.
from_file	Creates an image instance from an image file.
from_url	Creates an image instance from an image URL.

term_image.image.**auto_image_class**()

Selects the image render style that best suits the current terminal emulator.

Returns
An image class (a subclass of *BaseImage*).

Return type
term_image.image.common.ImageMeta

term_image.image.**AutoImage**(*image*, *, *width=None*, *height=None*, *scale=(1.0, 1.0)*)

Creates an image instance from a PIL image instance.

Returns
An instance of the automatically selected image render style (as returned by *auto_image_class()*).

Return type
term_image.image.common.BaseImage

Same arguments and raised exceptions as the *BaseImage* class constructor.

term_image.image.**from_file**(*filepath*, **kwargs*)

Creates an image instance from an image file.

Returns
An instance of the automatically selected image render style (as returned by *auto_image_class()*).

Return type
term_image.image.common.BaseImage

Same arguments and raised exceptions as *BaseImage.from_file()*.

term_image.image.**from_url**(*url*, **kwargs*)

Creates an image instance from an image URL.

Returns
An instance of the automatically selected image render style (as returned by *auto_image_class()*).

Return type
term_image.image.common.BaseImage

Same arguments and raised exceptions as *BaseImage.from_url()*.

Enumerations

ImageSource	Image source type.
Size	Enumeration for *automatic sizing*.

class term_image.image.**ImageSource**(*value*)

Bases: enum.Enum

Image source type.

FILE_PATH

The instance was derived from a path to a local image file.

PIL_IMAGE

The instance was derived from a PIL image instance.

URL

The instance was derived from an image URL.

class term_image.image.**Size**(*value*)

Bases: enum.Enum

Enumeration for *automatic sizing*.

AUTO

Equivalent to *ORIGINAL* if it will fit into the *available size*, else *FIT*.

FIT

The image size is set to fit optimally **within** the *available size*.

FIT_TO_WIDTH

The size is set such that the width is exactly the *available width*, regardless of the *cell ratio*.

ORIGINAL

The image size is set such that the image is rendered with as many pixels as the the original image consists of.

Image Classes

Class Hierachy

- *ImageMeta*
- *BaseImage*
  - *TextImage*
    - *BlockImage*
  - *GraphicsImage*
    - *ITerm2Image*
    - *KittyImage*

The Classes

ImageMeta	Type of all render style classes.
BaseImage	Base of all render styles.
TextImage	Base of all *Text-based Render Styles*.
BlockImage	A render style using unicode half blocks and 24-bit colour escape codes.
GraphicsImage	Base of all *Graphics-based Render Styles*.
ITerm2Image	A render style using the iTerm2 inline image protocol.
KittyImage	A render style using the Kitty terminal graphics protocol.

class term_image.image.**ImageMeta**(*name*, *bases*, *dict*, ***kwds*)

Bases: term_image.utils.ClassPropertyMeta, abc.ABCMeta

Type of all render style classes.

Note:

For all render style classes (instances of this class) defined **within** this package, `str(cls)` yields the same value as *cls.style*.

For render style classes defined **outside** this package (subclasses of those defined within this package), `str(cls)` is equivalent to `repr(cls)`.

Instance Properties:

style	Name of the render style [category].

property style

Name of the render style [category].

Returns

- The name of the render style [category] implemented by the invoking class, if defined **within** this package (term_image)
- None, if the invoking class is defined **outside** this package (term_image)

Type
Optional[str]

Examples

For a class defined within this package:

```
>>> from term_image.image import KittyImage
>>> KittyImage.style
'kitty'
```

For a class defined outside this package:

```
>>> from term_image.image import KittyImage
>>> class MyImage(KittyImage): pass
>>> MyImage.style is None
True
```

Hint: Equivalent to `str(cls)` for all render style classes (instances of *ImageMeta*) defined **within** this package.

---

class term_image.image.**BaseImage**(*image*, *, *width=None*, *height=None*, *scale=(1.0, 1.0)*)

Bases: object

Base of all render styles.

Parameters

- **image** (*PIL.Image.Image*) – Source image.
- **width** (*Union[int, Size, None]*) – Can be
  - an integer; horizontal dimension of the image, in columns.
  - a *Size* enum member.
- **height** (*Union[int, Size, None]*) – Can be
  - an integer; vertical dimension of the image, in lines.
  - a *Size* enum member.
- **scale** (*Tuple[float, float]*) – The fraction of the size (on respective axes) to render the image with.

Raises

- **TypeError** – An argument is of an inappropriate type.
- **ValueError** – An argument is of an appropriate type but has an unexpected/invalid value.

Propagates exceptions raised by *set_size()*, if *width* or *height* is given.

Note:

- If neither *width* nor *height* is given (or both are `None`), *FIT* applies.

- The image size is multiplied by the *scale* on respective axes when the image is *rendered*.
- For animated images, the seek position is initialized to the current seek position of the given image.
- It's allowed to set properties for *animated* images on non-animated ones, the values are simply ignored.

Attention: This class cannot be directly instantiated. Image instances should be created from its subclasses.

Instance Properties:

closed	Instance finalization status
frame_duration	Duration of a single frame for *animated* images
height	The **unscaled** height of the image
is_animated	`True` if the image is *animated*.
original_size	Size of the source image (in pixels)
n_frames	The number of frames in the image
rendered_height	The **scaled** height of the image
rendered_size	The **scaled** size of the image
rendered_width	The **scaled** width of the image
scale	Image *scale*
scale_x	Horizontal *scale*
scale_y	Vertical *scale*
size	The **unscaled** size of the image
source	The *source* from which the instance was initialized
source_type	The kind of *source* from which the instance was initialized
width	The **unscaled** width of the image

Class Properties:

forced_support	Render style forced support status

Instance Methods:

close	Finalizes the instance and releases external resources.
draw	Draws an image to standard output.
seek	Changes current image frame.
set_size	Sets the image size with extended control.
tell	Returns the current image frame number.

Class Methods:

from_file	Creates an instance from an image file.
from_url	Creates an instance from an image URL.
is_supported	Checks if the implemented *render style* is supported by the *active terminal*.

Class/Instance Methods:

`set_render_method`	Sets the *render method* used by instances of a *render style* class that implements multiple render methods.

property closed

Instance finalization status

Type
bool

property forced_support

Render style forced support status

Type
bool

GET:
Returns the forced support status of the render style of the invoker.

SET:
Forced support is enabled or disabled for the render style of the invoker.

If forced support is:

- **enabled**, the render style is treated as if it were supported, regardless of the return value of `is_supported()`.
- **disabled**, the return value of `is_supported()` determines if the render style is supported or not.

By **default**, forced support is **disabled** by the base style class (`BaseImage`).

Note:

- This property is *descendant*.
- This doesn't affect the return value of `is_supported()` but may affect operations that require that a render style be supported e.g instantiation of some render style classes.

---

property frame_duration

Duration of a single frame for *animated* images

Returns

- A duration (in seconds), if the image is animated.
- `None`, if the image is not animated.

Type
Optional[float]

Setting this on non-animated images is simply ignored.

property height

The **unscaled** height of the image

Returns

- The image height (in lines), if the image size is *fixed*.
- A `Size` enum member; if the image size is *dynamic*.

Type
Union[*Size*, int]

SETTABLE VALUES:

- a positive `int`; the image height is set to the given value and the width is set proportionally.
- a *`Size`* enum member; the image size is set as prescibed by the enum member.
- `None`; equivalent to *`FIT`*.

Setting this

- results in a *fixed size*.
- resets the recognized advanced sizing options to their defaults.

property is_animated

`True` if the image is *animated*. Otherwise, `False`.

property original_size

Size of the source image (in pixels)

Type
Tuple[int, int]

property n_frames: int

The number of frames in the image

Type
int

property rendered_height

The **scaled** height of the image

Also the exact number of lines that the drawn image will occupy in a terminal.

Type
int

property rendered_size

The **scaled** size of the image

Also the exact number of columns and lines (respectively) that the drawn image will occupy in a terminal.

Type
Tuple[int, int]

property rendered_width

The **scaled** width of the image

Also the exact number of columns that the drawn image will occupy in a terminal.

Type
int

property scale

Image *scale*

SETTABLE VALUES:

- A *scale value*; sets both axes.
- A `tuple` of two *scale values*; sets `(x, y)` respectively.

A scale value is a `float` in the range **0.0 < value <= 1.0**.

Type
Tuple[float, float]

property scale_x

Horizontal *scale*

A scale value is a `float` in the range **0.0 < x <= 1.0**.

Type
float

property scale_y

Vertical *scale*

A scale value is a `float` in the range **0.0 < y <= 1.0**.

Type
float

property size

The **unscaled** size of the image

Returns

- The image size, `(columns, lines)`, if the image size is *fixed*.
- A `Size` enum member, if the image size is *dynamic*.

Type
Union[*Size*, Tuple[int, int]]

SETTABLE VALUES:

- A `Size` enum member; the image size is set as prescibed by the enum member.

Setting this

- implies *dynamic sizing* i.e the size is computed whenever the image is *rendered*.
- resets the recognized advanced sizing options to their defaults.

This is multiplied by the *scale* on respective axes when the image is *rendered*.

property source

The *source* from which the instance was initialized

Type
Union[PIL.Image.Image, str]

property source_type

The kind of *source* from which the instance was initialized

Type
ImageSource

property width

The **unscaled** width of the image

Returns

- The image width (in columns), if the image size is *fixed*.
- A `Size` enum member; if the image size is *dynamic*.

Type
Union[*Size*, int]

SETTABLE VALUES:

- a positive `int`; the image width is set to the given value and the height is set proportionally.
- a *`Size`* enum member; the image size is set as prescibed by the enum member.
- `None`; equivalent to *`FIT`*.

Setting this

- results in a *fixed size*.
- resets the recognized advanced sizing options to their defaults.

`close``()`

Finalizes the instance and releases external resources.

- In most cases, it's not neccesary to explicity call this method, as it's automatically called when the instance is garbage-collected.
- This method can be safely called mutiple times.
- If the instance was initialized with a PIL image, the PIL image is never finalized.

`draw`(*h_align=None*, *pad_width=None*, *v_align=None*, *pad_height=None*, *alpha=0.1568627450980392*, ***, *scroll=False*, *animate=True*, *repeat=-1*, *cached=100*, *check_size=True*, ***style*)

Draws an image to standard output.

Parameters

- **`h_align`** (*`Optional[str]`*) – Horizontal alignment ("left" / "<", "center" / "|" or "right" / ">"). Default: center.
- **`pad_width`** (*`Optional[int]`*) – Number of columns within which to align the image.
  - Excess columns are filled with spaces.
  - Must not be greater than the *available terminal width*.
  - Default: terminal width, minus horizontal allowance.
- **`v_align`** (*`Optional[str]`*) – Vertical alignment ("top"/"^", "middle"/"-" or "bottom"/"_"). Default: middle.
- **`pad_height`** (*`Optional[int]`*) – Number of lines within which to align the image.
  - Excess lines are filled with spaces.
  - Must not be greater than the *available terminal height*, **for animations**.
  - Default: terminal height, minus vertical allowance.
- **`alpha`** (*`Optional[float, str]`*) – Transparency setting.
  - If `None`, transparency is disabled (alpha channel is removed).
  - If a `float` (**0.0 <= x < 1.0**), specifies the alpha ratio **above** which pixels are taken as **opaque**. **(Applies to only text-based render styles)**.
  - If a string, specifies a color to replace transparent background with. Can be:
    - **"#"** -> The terminal's default background color (or black, if undetermined) is used.

* A hex color e.g `ffffff`, `7faa52`.

- **scroll** (*bool*) – Only applies to non-animations. If `True`, allows the image's *rendered height* to be greater than the *available terminal height*.
- **animate** (*bool*) – If `False`, disable animation i.e draw only the current frame of an animated image.
- **repeat** (*int*) – The number of times to go over all frames of an animated image. A negative value implies infinite repetition.
- **cached** (*Union[bool, int]*) – Determines if *rendered* frames of an animated image will be cached (for speed up of subsequent renders of the same frame) or not.
  - If `bool`, it directly sets if the frames will be cached or not.
  - If `int`, caching is enabled only if the framecount of the image is less than or equal to the given number.
- **check_size** (*bool*) – If `False`, does not perform size validation for non-animations.
- **style** (*Any*) – Style-specific render parameters. See each subclass for it's own usage.

Raises

- `TypeError` – An argument is of an inappropriate type.
- `ValueError` – An argument is of an appropriate type but has an unexpected/invalid value.
- `ValueError` – Unable to convert or resize image.
- ***term_image.exceptions.InvalidSizeError*** – The image's *rendered size* can not fit into the *available terminal size*.
- ***term_image.exceptions.StyleError*** – Unrecognized style-specific parameter(s).

- If *set_size()* was used to set the image size, the horizontal and vertical allowances (set when *set_size()* was called) are taken into consideration during size validation. If the size was set via another means or the size is *dynamic*, the default allowances apply.
- For **non-animations**, if the image size was set with *FIT_TO_WIDTH*, the image **height** is not validated and setting *scroll* is unnecessary.
- *animate*, *repeat* and *cached* apply to *animated* images only. They are simply ignored for non-animated images.
- For animations (i.e animated images with *animate* set to `True`):
  - *scroll* is ignored.
  - Image size and *padding height* are always validated, if set or given.
  - **with the exception of native animations provided by some render styles**.
- Animations, **by default**, are infinitely looped and can be terminated with `SIGINT` (`CTRL + C`), raising `KeyboardInterrupt`.

classmethod from_file(*filepath*, ***kwargs*)

Creates an instance from an image file.

Parameters

- **filepath** (*str | os.PathLike*) – Relative/Absolute path to an image file.
- **kwargs** (*None | int | Tuple[float, float]*) – Same keyword arguments as the class constructor.

Returns
A new instance.

Raises

- `TypeError` – *filepath* is of an inappropriate type.
- `FileNotFoundError` – The given path does not exist.
- `IsADirectoryError` – Propagated from from `PIL.Image.open()`.
- `PIL.UnidentifiedImageError` – Propagated from from `PIL.Image.open()`.

Return type
term_image.image.common.BaseImage

Also Propagates exceptions raised or propagated by the class constructor.

classmethod from_url(*url*, ***kwargs*)

Creates an instance from an image URL.

Parameters

- **url** (*str*) – URL of an image file.
- **kwargs** (*None | int | Tuple[float, float]*) – Same keyword arguments as the class constructor.

Returns
A new instance.

Raises

- `TypeError` – *url* is not a string.
- `ValueError` – The URL is invalid.
- *term_image.exceptions.URLNotFoundError* – The URL does not exist.
- `PIL.UnidentifiedImageError` – Propagated from `PIL.Image.open()`.

Return type
term_image.image.common.BaseImage

Also propagates connection-related exceptions from `requests.get()` and exceptions raised or propagated by the class constructor.

Note: This method creates a temporary file, but only after successful initialization. The file is removed:

- when *close()* is called,
- upon exiting a `with` statement block that uses the instance as a context manager, or
- when the instance is garbage collected.

---

abstract classmethod is_supported()

Checks if the implemented *render style* is supported by the *active terminal*.

Returns
`True` if the render style implemented by the invoking class is supported by the *active terminal*. Otherwise, `False`.

Return type
bool

> **Attention:** Support checks for most (if not all) render styles require *querying* the *active terminal* the **first time** they're executed.
>
> Hence, it's advisable to perform all neccesary support checks (call this method on required style classes) at an early stage of a program, before user input is expected. If using automatic style selection, calling `auto_image_class()` only should be sufficient.

seek(*pos*)

Changes current image frame.

Parameters
pos (*int*) – New frame number.

Raises

- **TypeError** – An argument is of an inappropriate type.
- **ValueError** – An argument is of an appropriate type but has an unexpected/invalid value.

Frame numbers start from 0 (zero).

classmethod set_render_method(*method=None*)

Sets the *render method* used by instances of a *render style* class that implements multiple render methods.

Parameters
method (*str | None*) – The render method to be set or `None` for a reset (case-insensitive).

Raises

- **TypeError** – *method* is not a string or `None`.
- **ValueError** – the given method is not implmented by the invoking class (or class of the invoking instance).

See the **Render Methods** section in the description of subclasses that implement such for their specific usage.

If *method* is not `None` and this method is called via:

- a class, the class-wide render method is set.
- an instance, the instance-specific render method is set.

If *method* is `None` and this method is called via:

- a class, the class-wide render method is unset, so that it uses that of its parent style class (if any) or the default.
- an instance, the instance-specific render method is unset, so that it uses the class-wide render method thenceforth.

Any instance without a render method set uses the class-wide render method.

Note: *method* = `None` is always allowed, even if the render style doesn't implement multiple render methods.

The **class-wide** render method is *descendant*.

---

set_size(*width=None*, *height=None*, *h_allow=0*, *v_allow=2*, *maxsize=None*)

Sets the image size with extended control.

Parameters

- **width** (*int* | term_image.image.common.Size | *None*) – Can be
  - an integer; horizontal dimension of the image, in columns.
  - a *Size* enum member.
- **height** (*int* | term_image.image.common.Size | *None*) – Can be
  - an integer; vertical dimension of the image, in lines.
  - a *Size* enum member.
- **h_allow** (*int*) – Horizontal allowance i.e minimum number of columns to leave unused.
- **v_allow** (*int*) – Vertical allowance i.e minimum number of lines to leave unused.
- **maxsize** (*Tuple[int, int]* | *None*) – If given, as `(columns, lines)`, it's used instead of the terminal size.

Raises

- **TypeError** – An argument is of an inappropriate type.
- **ValueError** – An argument is of an appropriate type but has an unexpected/invalid value.
- **ValueError** – Both *width* and *height* are specified.
- **ValueError** – The *available size* is too small for *automatic sizing*.
- ***term_image.exceptions.InvalidSizeError*** – *maxsize* is given and the resulting size will not fit into it.

If neither *width* nor *height* is given (or both are `None`), *FIT* applies.

If *width* or *height* is a *Size* enum member, *automatic sizing* applies as prescribed by the enum member.

When *FIT_TO_WIDTH* is given,

- size validation operations take it into consideration.
- *Vertical allowance* is nullified.

Allowances are ignored when *maxsize* is given.

Render formatting and size validation operations recognize and respect the horizontal and vertical allowances, until the image size is re-set.

Note: The size is checked to fit in only when *maxsize* is given along with a fixed *width* or *height* because *draw()* is generally not the means of drawing such an image and all rendering methods don't perform any sort of size validation.

If the validation is not desired, specify only one of *maxsize* and *width* or *height*, not both.

---

tell()

Returns the current image frame number.

Return type

int

class term_image.image.**TextImage**(*image*, *, *width=None*, *height=None*, *scale=(1.0, 1.0)*)

Bases: *term_image.image.common.BaseImage*

Base of all *Text-based Render Styles*.

See *BaseImage* for the description of the constructor.

Important: Instantiation of subclasses is always allowed, even if the current terminal does not [fully] support the render style.

To check if the render style is fully supported in the current terminal, use *is_supported()*.

Attention: This class cannot be directly instantiated. Image instances should be created from its subclasses.

class term_image.image.**BlockImage**(*image*, *, *width=None*, *height=None*, *scale=(1.0, 1.0)*)

Bases: *term_image.image.common.TextImage*

A render style using unicode half blocks and 24-bit colour escape codes.

See *TextImage* for the description of the constructor.

class term_image.image.**GraphicsImage**(*image*, *, *width=None*, *height=None*, *scale=(1.0, 1.0)*)

Bases: *term_image.image.common.BaseImage*

Base of all *Graphics-based Render Styles*.

Raises

term_image.exceptions.StyleError – The *active terminal* doesn't support the render style.

See *BaseImage* for the description of the constructor.

Attention: This class cannot be directly instantiated. Image instances should be created from its subclasses.

Tip: To allow instantiation regardless of whether the render style is supported or not, enable *forced_support*.

---

class term_image.image.**ITerm2Image**(*image*, *, *width=None*, *height=None*, *scale=(1.0, 1.0)*)

Bases: *term_image.image.common.GraphicsImage*

A render style using the iTerm2 inline image protocol.

See *GraphicsImage* for the complete description of the constructor.

Render Methods

ITerm2Image provides two methods of *rendering* images, namely:

LINES (default)
Renders an image line-by-line i.e the image is evenly split across the number of lines it should occupy.

Pros:

- Good for use cases where it might be required to trim some lines of the image.

Cons:

- Image drawing is very slow on iTerm2 due to the terminal emulator's performance.

WHOLE
Renders an image all at once i.e the entire image data is encoded into one line of the *rendered* output, such that the entire image is drawn once by the terminal and still occupies the correct amount of lines and columns.

Pros:

- Render results are more compact (i.e less in character count) than with the **LINES** method since the entire image is encoded at once.
- Image drawing is faster than with **LINES** on most terminals.
- Smoother animations.

Cons:

- This method currently doesn't work well on iTerm2 and WezTerm when the image height is greater than the terminal height.

Note: The **LINES** method is the default only because it works properly in all cases, it's more advisable to use the **WHOLE** method except when the image height is greater than the terminal height or when trimming the image is required.

The render method can be set with *set_render_method()* using the names specified above.

Style-Specific Render Parameters

See *BaseImage.draw()* (particularly the *style* parameter).

- **method** (*None* | *str*) → Render method override.
  - `None` → the current effective render method of the instance is used.
  - *default* → `None`
- **mix** (*bool*) → Cell content inter-mix policy (**Only supported on WezTerm**, ignored otherwise).
  - `False` → existing contents of cells within the region covered by the drawn render output are erased
  - `True` → existing cell contents show under transparent areas of the drawn render output
  - *default* → `False`
- **compress** (*int*) → ZLIB compression level, for renders re-encoded in PNG format.
  - `0` <= *compress* <= `9`
  - `1` → best speed, `9` → best compression, `0` → no compression
  - *default* → `4`
  - Results in a trade-off between render time and data size/draw speed
- **native** (*bool*) → Native animation policy.[1]
  - `True` → use the protocol's native animation feature
  - `False` → use the normal animation
  - *default* → `False`
  - *alpha*, *repeat*, *cached* and *style* do not apply
  - Ignored if the image is not animated or *animate* is `False`
  - Normal restrictions for sizing of animations do not apply
  - Uses **WHOLE** render method
  - The terminal emulator completely controls the animation
- **stall_native** (*bool*) → Native animation execution control.
  - `True` → block until `SIGINT` (Ctrl+C) is recieved
  - `False` → return as soon as the image is transmitted
  - *default* → `True`

Format Specification

See *Render Format Specification*.

```
[ <method> ]  [ m <mix> ]  [ c <compress> ]
```

- `method` → render method override

[1] Native animation support:
- Not all animated image formats may be supported by every supported terminal emulator
- Not all supported terminal emulators implement this feature of the protocol e.g on Konsole, the first frame is drawn but the image is not animated

  - `L` → **LINES** render method (current frame only, for animated images)
  - `W` → **WHOLE** render method (current frame only, for animated images)
  - `N` → Native animation[Page 38, 1] (ignored when used with non-animated images or *`ImageIterator`*)
  - *default* → current effective render method of the instance
- `m` → cell content inter-mix policy (**Only supported in WezTerm**, ignored otherwise)
  - `mix` → inter-mix policy
    - `0` → existing contents of cells in the region covered by the drawn render output will be erased
    - `1` → existing cell contents show under transparent areas of the drawn render output
  - *default* → `m0`
  - e.g `m0`, `m1`
- `c` → ZLIB compression level, for renders re-encoded in PNG format
  - `compress` → compression level
    - An integer in the range `0` <= `x` <= `9`
    - `1` → best speed, `9` → best compression, `0` → no compression
  - *default* → `c4`
  - e.g `c0`, `c9`
  - Results in a trade-off between render time and data size/draw speed

---

Important: Currently supported terminal emulators are:

- iTerm2
- Konsole >= 22.04.0
- WezTerm

Class/Instance Properties:

`jpeg_quality`	JPEG encoding quality
`read_from_file`	Read-from-file optimization policy

Class Properties:

`native_anim_max_bytes`	Maximum size (in bytes) of image data for native animation

Class Methods:

clear	Clears images.

`property jpeg_quality`

JPEG encoding quality

Type
int

GET:
Returns the effective JPEG encoding quality of the invoker (negative if disabled).

SET:
If invoked via:

- a **class**, the **class-wide** quality is set.
- an **instance**, the **instance-specific** quality is set.

DELETE:
If invoked via:

- a **class**, the **class-wide** quality is unset.
- an **instance**, the **instance-specific** quality is unset.

If:

- *value* < `0`; JPEG encoding is disabled.
- `0` <= *value* <= `95`; JPEG encoding is enabled with the given quality.

If **unset** for:

- a **class**, it uses that of its parent *iterm2* style class (if any) or the default (disabled), if unset for all parents or the class has no parent *iterm2* style class.
- an **instance**, it uses that of it's class.

By **default**, the quality is **unset** i.e JPEG encoding is **disabled** and images are encoded in the PNG format (when not reading directly from file) but in some cases, higher and/or faster compression may be desired. JPEG encoding is significantly faster than PNG encoding and produces smaller (in data size) output but **at the cost of image quality**.

Note:

- This property is *descendant*.
- This optimization applies to only **re-encoded** (i.e not read directly from file) **non-transparent** renders.

Tip: The transparency status of some images can not be correctly determined in an efficient way at render time. To ensure JPEG encoding is always used for a re-encoded render, disable transparency or set a background color.

Furthermore, to ensure that renders with the **WHOLE** *render method* are always re-encoded, disable `read_from_file`.

This optimization is useful in improving non-native animation performance.

See also:

- the *alpha* parameter of `draw()` and the `#`, `bgcolor` fields of the *Render Format Specification*
- `read_from_file`

property native_anim_max_bytes

Maximum size (in bytes) of image data for native animation

Type
int

GET:
Returns the set value.

SET:
A positive integer; the value is set on the *iterm2* render style baseclass (`ITerm2Image`).

DELETE:
The value is unset, thereby resetting it to the default.

`TermImageWarning` is issued (and shown **only the first time**, except a filter is set to do otherwise) if the image data size for a native animation is above this value.

Note: This property is *descendant* but is always unset for all subclasses and instances. Hence, setting/resetting it on this class, a subclass or an instance affects this class, all its subclasses and all their instances.

Warning: This property should be altered with caution to avoid excessive memory usage.

property read_from_file

Read-from-file optimization policy

Type
bool

GET:
Returns the effective read-from-file policy of the invoker.

SET:
If invoked via:

- a **class**, the **class-wide** policy is set.
- an **instance**, the **instance-specific** policy is set.

DELETE:
If invoked via:

- a **class**, the **class-wide** policy is unset.

- an **instance**, the **instance-specific** policy is unset.

If the value is:

- `True`, image data is read directly from file when possible and no image manipulation is required.
- `False`, images are always re-encoded (in the PNG format by default).

If **unset** for:

- a **class**, it uses that of its parent *iterm2* style class (if any) or the default (`True`), if unset for all parents or the class has no parent *iterm2* style class.
- an **instance**, it uses that of it's class.

By **default**, the policy is **unset**, which is equivalent to `True` i.e the optimization is **enabled**.

Note:

- This property is *descendant*.
- This is an optimization to reduce render times and is only applicable to the **WHOLE** render method, since the the **LINES** method inherently requires image manipulation.
- This property does not affect animations. Native animations are always read from file when possible and frames of non-native animations have to be re-encoded.

See also:

jpeg_quality

classmethod clear(*cursor=False*, *now=False*)

Clears images.

Parameters

- **cursor** (*bool*) – If `True`, all images intersecting with the current cursor position are cleared. Otherwise, all visible images are cleared.
- **now** (*bool*) – If `True` the images are cleared immediately, without affecting any standard I/O stream. Otherwise they're cleared when next `sys.stdout` is flushed.

Note: Required and works only on Konsole, as text doesn't overwrite images.

---

class term_image.image.**KittyImage**(*image*, *, *width=None*, *height=None*, *scale=(1.0, 1.0)*)

Bases: *term_image.image.common.GraphicsImage*

A render style using the Kitty terminal graphics protocol.

See *GraphicsImage* for the complete description of the constructor.

Render Methods

`KittyImage` provides two methods of *rendering* images, namely:

LINES (default)
: Renders an image line-by-line i.e the image is evenly split across the number of lines it should occupy.

  Pros:

  - Good for use cases where it might be required to trim some lines of the image.

WHOLE
: Renders an image all at once i.e the entire image data is encoded into one line of the *rendered* output, such that the entire image is drawn once by the terminal and still occupies the correct amount of lines and columns.

  Pros:

  - Render results are more compact (i.e less in character count) than with the **LINES** method since the entire image is encoded at once.

The render method can be set with `set_render_method()` using the names specified above.

Style-Specific Render Parameters

See `BaseImage.draw()` (particularly the *style* parameter).

- **method** (*None* | *str*) → Render method override.
  - `None` → the current effective render method of the instance is used.
  - *default* → `None`
- **z_index** (*int*) → The stacking order of graphics and text for **non-animations**.
  - An integer in the **signed 32-bit** range (excluding `-(2**31)`)
  - `>= 0` → the image will be drawn above text
  - `< 0` → the image will be drawn below text
  - `< -(2**31)/2` → the image will be drawn below cells with non-default background color
  - *default* → `0`
  - Overlapping graphics on different z-indexes will be blended (by the terminal emulator) if they are semi-transparent.
  - To inter-mix text with graphics, see the *mix* parameter.
- **mix** (*bool*) → Graphics/Text inter-mix policy.
  - `False` → text within the region covered by the drawn render output will be erased, though text can be inter-mixed with graphics after drawing
  - `True` → text within the region covered by the drawn render output will NOT be erased
  - *default* → `False`
- **compress** (*int*) → ZLIB compression level.
  - `0 <= compress <= 9`

  - `1` → best speed, `9` → best compression, `0` → no compression
  - *default* → `4`
  - Results in a trade-off between render time and data size/draw speed

Format Specification

See *Render Format Specification*.

```
[ <method> ]  [ z <z-index> ]  [ m <mix> ]  [ c <compress> ]
```

- `method` → render method override
  - `L` → **LINES** render method (current frame only, for animated images)
  - `W` → **WHOLE** render method (current frame only, for animated images)
  - *default* → Current effective render method of the image
- `z` → graphics/text stacking order
  - `z-index` → z-index
    - An integer in the **signed 32-bit** range (excluding `-(2**31)`)
    - `>= 0` → the render output will be drawn above text
    - `< 0` → the render output will be drawn below text
    - `< -(2**31)/2` → the render output will be drawn below cells with non-default background color
  - *default* → `z0` (z-index zero)
  - e.g `z0`, `z1`, `z-1`, `z2147483647`, `z-2147483648`
  - overlapping graphics on different z-indexes will be blended (by the terminal emulator) if they are semi-transparent
- `m` → graphics/text inter-mix policy
  - `mix` → inter-mix policy
    - `0` → text within the region covered by the drawn render output will be erased, though text can be inter-mixed with graphics after drawing
    - `1` → text within the region covered by the drawn render output will NOT be erased
  - *default* → `m0`
  - e.g `m0`, `m1`
- `c` → ZLIB compression level
  - `compress` → compression level
    - An integer in the range `0 <= compress <= 9`
    - `1` → best speed, `9` → best compression, `0` → no compression
  - *default* → `c4`

  - e.g `c0`, `c9`
  - results in a trade-off between render time and data size/draw speed

Important: Currently supported terminal emulators are:

- Kitty >= 0.20.0.
- Konsole >= 22.04.0.

Class Methods:

clear	Clears images.

classmethod clear(*, *cursor=False*, *z_index=None*, *now=False*)

Clears images.

Parameters

- **cursor** (*bool*) – If `True`, all images intersecting with the current cursor position are cleared.
- **z_index** (*int | None*) – An integer in the **signed 32-bit range**. If given, all images on the given z-index are cleared.
- **now** (*bool*) – If `True` the images are cleared immediately, without affecting any standard I/O stream. Otherwise they're cleared when next `sys.stdout` is flushed.

Aside *now*, **only one** other argument may be given. If no argument is given (aside *now*) or default values are given, all images visible on the screen are cleared.

Note: This method does nothing if the render style is not supported.

Context Management Protocol Support

BaseImage instances are context managers i.e they can be used with the `with` statement as in:

```
with from_url(url) as image:
    ...
```

Using an instance as a context manager guarantees **instant** object **finalization** (i.e clean-up/release of resources), especially for instances with URL sources (see *BaseImage.from_url()*).

Iteration Support

Animated images are iterable i.e they can be used with the `for` statement (and other means of iteration such as unpacking) as in:

```
for frame in from_file("animated.gif"):
    ...
```

Subsequent frames of the image are yielded on subsequent iterations.

Note:

- `iter(anim_image)` returns an *`ImageIterator`* instance with a repeat count of `1`, hence caching is disabled.
- The frames are unformatted and transparency is enabled i.e as returned by `str(image)`.

For extensive or custom iteration, use *`ImageIterator`* directly.

Other Classes

class term_image.image.**ImageIterator**(*image*, *repeat=-1*, *format_spec=''*, *cached=100*)

Bases: `object`

Effeciently iterate over *rendered* frames of an *animated* image

Parameters

- **image** (`BaseImage`) – Animated image.
- **repeat** (*`int`*) – The number of times to go over the entire image. A negative value implies infinite repetition.
- **format_spec** (*`str`*) – The *format specifier* for the rendered frames (default: auto).
- **cached** (*`Union[bool, int]`*) – Determines if the *rendered* frames will be cached (for speed up of subsequent renders) or not. If it is
  - a boolean, it directly sets if the frames will be cached or not.
  - an integer, caching is enabled only if the framecount of the image is less than or equal to the given number.

Raises

- `TypeError` – An argument is of an inappropriate type.
- `ValueError` – An argument is of an appropriate type but has an unexpected/invalid value.
- *`term_image.exceptions.StyleError`* – Invalid style-specific format specifier.

- If *repeat* equals `1`, caching is disabled.
- The iterator has immediate response to changes in the image size and *scale*.
- If the image size is *dynamic*, it's computed per frame.
- The number of the last yielded frame is set as the image's seek position.

- Directly adjusting the seek position of the image doesn't affect iteration. Use *ImageIterator.seek()* instead.
- After the iterator is exhausted, the underlying image is set to frame `0`.

Instance Properties:

loop_no	Iteration repeat countdown

Instance Methods:

close	Closes the iterator and releases resources used.
seek	Sets the frame number to be yielded on the next iteration without affecting the repeat count.

property loop_no

Iteration repeat countdown

Type
int

Changes on the first iteration of each loop, except for infinite iteration where it's always `-1`.

close()

Closes the iterator and releases resources used.

Does not reset the frame number of the underlying image.

Note: This method is automatically called when the iterator is exhausted or garbage-collected.

seek(*pos*)

Sets the frame number to be yielded on the next iteration without affecting the repeat count.

Parameters
pos (*int*) – Next frame number.

Raises

- **TypeError** – An argument is of an inappropriate type.
- **ValueError** – An argument is of an appropriate type but has an unexpected/invalid value.
- ***term_image.exceptions.TermImageError*** – Iteration has not yet started or the iterator is exhausted/closed.

Frame numbers start from `0` (zero).

1.3.3 `widget` Module

Classes:

UrwidImage	Image widget (box/flow) for the urwid TUI framework.
UrwidImageCanvas	Image canvas for the urwid TUI framework.
UrwidImageScreen	A screen that supports drawing images.

class term_image.widget.**UrwidImage**(*image*, *format_spec=''*, ***, *upscale=False*)

Bases: urwid.widget.Widget

Image widget (box/flow) for the urwid TUI framework.

Parameters

- **image** (BaseImage) – The image to be rendered by the widget.
- **format_spec** (*str*) – *Render format specifier*. Padding width and height are ignored.
- **upscale** (*bool*) – If True, the image will be upscaled to fit maximally within the available size, if neccessary, while still preserving the aspect ratio. Otherwise, the image is never upscaled.

Raises

- **TypeError** – An argument is of an inappropriate type.
- **ValueError** – An argument is of an appropriate type but has an unexpected/invalid value.
- ***term_image.exceptions.StyleError*** – Invalid style-specific format specifier.
- ***term_image.exceptions.UrwidImageError*** – Too many image widgets rendering images with the *kitty* render style.

Any ample space in the widget's render size is filled with spaces.
For animated images, the current frame (at render-time) is rendered.

Tip: If *image* is of a *graphics-based* render style and the widget is being used as or within a **flow** widget, with overlays or in any other case where the canvas will require vertical trimming, make sure to use a render method that splits images across lines such as the **LINES** render method for *kitty* and *iterm2* render styles.

Note:

- The *z-index* style-specific format spec field for *KittyImage* is ignored as this is used internally.
- A **maximum** of `2**32 - 2` instances initialized with *KittyImage* instances may exist at the same time.

Important: This is defined if and only if the `urwid` package is available.

Instance Properties:

image	The image rendered by the widget.

Instance Methods:

clear	Clears all images drawn by the widget, if the image rendered by the widget is of the *kitty* render style.

Static Methods:

clear_all	Clears all on-screen images of *graphics-based* styles that support/require such an operation.

Class Methods:

set_error_placeholder	Sets the widget to be rendered in place of an image when rendering fails.

property image

The image rendered by the widget.

Type
BaseImage

clear(*, *now=False*)

Clears all images drawn by the widget, if the image rendered by the widget is of the *kitty* render style.

Parameters
now (*bool*) – If `True` the images are cleared immediately. Otherwise they're cleared just before the next screen redraw.

static clear_all(*, *now=False*)

Clears all on-screen images of *graphics-based* styles that support/require such an operation.

Parameters
now (*bool*) – If `True` the images are cleared immediately. Otherwise they're cleared just before the next screen redraw.

classmethod set_error_placeholder(*widget*)

Sets the widget to be rendered in place of an image when rendering fails.

Parameters
widget (*urwid.widget.Widget | None*) – The placholder widget or `None` to remove the placeholder.

Raises
TypeError – *widget* is not an urwid widget.

If set, any exception raised during rendering is **suppressed** and the placeholder is rendered in place of the image.

class term_image.widget.**UrwidImageCanvas**(*render*, *size*, *image_size*)

Bases: `urwid.canvas.Canvas`

Image canvas for the urwid TUI framework.

Parameters

- **render** (*str*) – The rendered image.
- **size** (*Tuple[int, int]*) – The canvas size. Also, the size of the rendered (and formatted) image.
- **image_size** (*Tuple[int, int]*) – The size with which the image was rendered (excluding padding).

Note: The canvas outputs blanks (spaces) for *graphics-based* images when horizontal trimming is required (e.g when a widget is laid over an image). This is temporary as horizontal trimming will be implemented in the future.

This canvas is intended to be rendered by *UrwidImage* (or a sublass of it) only. Otherwise, the output isn't guaranteed to be as expected.

Warning: The constructor of this class performs NO argument validation at all for the sake of performance. If instantiating this class directly, make sure to pass appropriate arguments or create subclass, override the constructor and perform the validation.

Important: This is defined if and only if the `urwid` package is available.

---

class term_image.widget.**UrwidImageScreen**(**args*, ***kwargs*)

Bases: urwid.raw_display.Screen

A screen that supports drawing images.

It monitors images of some *graphics-based* render styles and clears them off the screen when necessary (e.g at startup, when scrolling, upon terminal resize and at exit).

See the baseclass for further description.

Important: This is defined if and only if the `urwid` package is available.

Instance Methods:

draw_screen	See the description of the baseclass' method.
flush	See the baseclass' method for the description.
get_available_raw_input	See the baseclass' method for the description.
write	See the baseclass' method for the description.

draw_screen(*maxres*, *canvas*)

See the description of the baseclass' method.

Synchronizes output on terminal emulators that support the feature to reduce/eliminate image flickering and screen tearing.

Important: Synchronized with *term_image.utils.lock_tty()*.

---

flush()

See the baseclass' method for the description.

Important: Synchronized with *term_image.utils.lock_tty()*.

get_available_raw_input()

See the baseclass' method for the description.

Important: Synchronized with *term_image.utils.lock_tty()*.

write(*data*)

See the baseclass' method for the description.

Important: Synchronized with *term_image.utils.lock_tty()*.

1.3.4 exceptions Module

Warnings:

TermImageWarning	Package-specific warning category.

Exceptions:

TermImageError	Exception baseclass.
URLNotFoundError	Raised for 404 errors.
InvalidSizeError	Raised for invalid image sizes.
StyleError	Baseclass of style-specific exceptions.
GraphicsImageError	Raised for errors specific to *GraphicsImage* and its subclasses defined outside this package.
TextImageError	Raised for errors specific to *TextImage* and its subclasses defined outside this package.
BlockImageError	Raised for errors specific to *BlockImage* and its subclasses defined outside this package.
ITerm2ImageError	Raised for errors specific to *ITerm2Image* and its subclasses defined outside this package.
KittyImageError	Raised for errors specific to *KittyImage* and its subclasses defined outside this package.
UrwidImageError	Raised for errors specific to *UrwidImage*.

exception term_image.exceptions.**TermImageWarning**

Bases: UserWarning

Package-specific warning category.

exception term_image.exceptions.**TermImageError**

Bases: Exception

Exception baseclass. Raised for generic errors.

exception term_image.exceptions.**URLNotFoundError**

Bases: FileNotFoundError, *term_image.exceptions.TermImageError*

Raised for 404 errors.

exception term_image.exceptions.**InvalidSizeError**

Bases: ValueError, *term_image.exceptions.TermImageError*

Raised for invalid image sizes.

exception term_image.exceptions.**StyleError**

Bases: *term_image.exceptions.TermImageError*

Baseclass of style-specific exceptions.

Never raised for errors pertaining to image classes defined in this package. Instead, the exception subclass specific to each image class is raised.

Only raised for subclasses of *BaseImage* defined outside this package (which are not subclasses of any other image class defined in this package).

Being the baseclass of all style-specific exceptions, it can be used be used to handle any style-specific error, regardless of the render style it originated from.

exception term_image.exceptions.**GraphicsImageError**

Bases: *term_image.exceptions.StyleError*

Raised for errors specific to *GraphicsImage* and its subclasses defined outside this package.

exception term_image.exceptions.**TextImageError**

Bases: *term_image.exceptions.StyleError*

Raised for errors specific to *TextImage* and its subclasses defined outside this package.

exception term_image.exceptions.**BlockImageError**

Bases: *term_image.exceptions.TextImageError*

Raised for errors specific to *BlockImage* and its subclasses defined outside this package.

exception term_image.exceptions.**ITerm2ImageError**

Bases: *term_image.exceptions.GraphicsImageError*

Raised for errors specific to *ITerm2Image* and its subclasses defined outside this package.

exception term_image.exceptions.**KittyImageError**

Bases: *term_image.exceptions.GraphicsImageError*

Raised for errors specific to *KittyImage* and its subclasses defined outside this package.

exception term_image.exceptions.**UrwidImageError**

Bases: *term_image.exceptions.TermImageError*

Raised for errors specific to *UrwidImage*.

1.3.5 `utils` Module

Functions:

get_terminal_name_version	Returns the name and version of the *active terminal*, if available.
get_terminal_size	Returns the current size of the *active terminal*.
lock_tty	Synchronizes access to the *active terminal*.
read_tty_all	Reads all available input directly from the *active terminal* **without blocking**.
write_tty	Writes to the *active terminal* and waits until complete transmission.

`term_image.utils.`**`get_terminal_name_version`**`()`

Returns the name and version of the *active terminal*, if available.

Returns
A 2-tuple, `(name, version)`. If either is not available, returns `None` in its place.

Return type
Tuple[str | None, str | None]

`term_image.utils.`**`get_terminal_size`**`()`

Returns the current size of the *active terminal*.

Returns
The terminal size in columns and lines.

Return type
os.terminal_size

Note: This implementation is quite different from `shutil.get_terminal_size()` and `os.get_terminal_size()` in that it:

- gives the correct size of the *active terminal* even when output is redirected, in most cases
- gives different results in certain situations
- is what this library works with

`term_image.utils.`**`lock_tty`**`(`*`func`*`)`

Synchronizes access to the *active terminal*.

Parameters
func (*function*) – The function to be wrapped.

When a decorated function is called, a re-entrant lock is acquired by the current process or thread and released after the call, such that any other decorated function called within another thread or subprocess waits until the lock is fully released (i.e has been released as many times as acquired) by the current process or thread.

Note: It works across parent-/sub-processes, started directly or indirectly via `multiprocessing.Process` (or a subclass of it), and their threads, provided `multiprocessing.synchronize` is supported on the host platform. Otherwise, a warning is issued when starting a subprocess.

> **Warning:** If `multiprocessing.synchronize` is supported and a subprocess is started within a call (possibly recursive) to a decorated function, the thread in which that occurs will be out of sync until that call returns. Hence, avoid starting a subprocess within a decorated function.

`term_image.utils.`**`read_tty_all`**`()`

Reads all available input directly from the *active terminal* **without blocking**.

Returns
The input read.

Return type
bytes

Important: Synchronized with *`term_image.utils.lock_tty()`*.

Note: Currently works on UNIX only, returns `None` on any other platform or when there is no *active terminal*.

`term_image.utils.`**`write_tty`**`(`*`data`*`)`

Writes to the *active terminal* and waits until complete transmission.

Parameters
`data` (*`bytes`*) – Data to be written.

Important: Synchronized with *`term_image.utils.lock_tty()`*.

Note: Currently works on UNIX only, returns `None` on any other platform or when there is no *active terminal*.

1.4 Planned Features

In no particular order:

- Support for more terminal graphics protocols (See #23)
- More text-based render styles (See #57)
- Support for terminal emulators with less colors (See #61)
- Support for terminal emulators without Unicode support (See #58, #60)
- Support for fbTerm
- Support for open file objects
- Determination of frame duration per frame during animations and image iteration
- Asynchronous animation rendering
- Kitty image ID (See #40)
- Kitty native animation (See #40)
- Image zoom and pan functionalities

- Image trimming
- Specify key to end animation
- Drawing images to an alternate output
- Source images from raw pixel data
- IPython Extension
- and much more...

1.5 Known Issues

1. Drawing of images and animations doesn't work completely well with Python for Windows (tested in Windows Terminal and MinTTY).

   - **Description:** Some lines of the image seem to extend beyond the number of columns that they should normally occupy by one or two columns.

     This behaviour causes animations to go bizzare when lines extend beyond the width of the terminal emulator.

   - **Comment:** First of all, the issue seems to caused by the layer between Python and the terminal emulators (i.e the PTY implementation in use) which "consumes" the escape sequences used to display images.

     It is neither a fault of this library nor of the terminal emulators, as drawing of images and animations works properly with WSL within Windows Terminal.

   - **Solution:** A workaround is to leave some *horizontal allowance* of **at least two columns** to ensure the image never reaches the right edge of the terminal.

     This can be achieved in the library using the *h_allow* parameter of `set_size()`.

2. Animations with the **kitty** render style on the **Kitty terminal emulator** might be glitchy for some images with **high resolution and size** and/or **sparse color distribution**.

   - **Description:** When the **LINES** render method is used, lines of the image might intermittently disappear. When the **WHOLE** render method is used, the entire image might intermitently dissapear.
   - **Comment:** This is due to the fact that drawing each frame requires clearing the previous frame off the screen, since the terminal would otherwise blend subsequent frames. Not clearing previous frames would break transparent animations and result in a performance lag that gets worse over time.
   - **Solution:** Plans are in motion to implement support for native animations i.e utilizing the animation features provided by the protocol (See #40).

1.6 FAQs

Why?

- Why not?
- To improve and extend the capabilities of CLI and TUI applications.
- Terminals emulators have always been and always will be!

What about Windows support?

- Only the new Windows Terminal seems to have proper ANSI support and mordern terminal emulator features.
- Drawing images and animations doesn't work completely well with Python for Windows. See *Known Issues*.
- If stuck on Windows and want to use all features, you could use WSL + Windows Terminal.

Why are colours not properly reproduced?

- Some terminals support 24-bit colors but have a **256-color pallete**. This limits color reproduction.

Why are images out of scale?

- If *Auto Cell Ratio* is supported and enabled, call `enable_win_size_swap()`. If this doesn't work, then open an issue here with adequate details.
- Otherwise, adjust the *cell ratio* using `set_cell_ratio()`.

Why does my program get garbage input (possibly also written to the screen) or phantom keystrokes?

- This is most definitely due to slow reponse of the terminal emulator to *Terminal Queries*.
- To resolve this, set a higher timeout using `set_query_timeout()`. The default is `DEFAULT_QUERY_TIMEOUT` seconds.
- You can also disable terminal queries using `disable_queries()` but note that this disables certain *features*.

1.7 Glossary

Below are definitions of terms used across the API, exception messages and the documentation.

Note: For contributors, some of these terms are also used in the source code, as variable names, in comments, docstrings, etc.

---

active terminal
The terminal emulator connected to the first TTY device discovered upon loading the `term_image` package.

At times, this may also be used to refer to the TTY device itself.

See also:

The Active Terminal

alignment
The position of a primary *render* output within its *padding*.

See also:

Alignment

allowance
The amount of space to be left unused on the terminal screen.

alpha threshold
Alpha ratio/value above which a pixel is taken as **opaque** (applies only to *Text-based Render Styles*).

See also:

Transparency

animated
: Having multiple frames.

  The frames of an animated image are generally meant to be displayed in rapid succession, to give the effect of animation.

automatic size
automatic sizing
: A form of sizing wherein the image size is computed based on a combination of the *available size*, the image's original size and a given width **or** height.

  This form of sizing tries to preserve image aspect ratio and can be used with both *fixed sizing* and *dynamic sizing*.

  See also:

  manual sizing, `Size` and `set_size()`

available height
: The remainder after vertical allowance is subtracted from the maximum amount of lines.

available size
: The remainder after *allowances* are subtracted from the maximum frame size.

available width
: The remainder after horizontal allowance is subtracted from the maximum amount of columns.

cell ratio
: The **aspect ratio** (i.e the ratio of **width to height**) of a **character cell** on a terminal screen.

  See also:

  `get_cell_ratio()` and `set_cell_ratio()`

descendant
: Refers to an attribute, property or setting set on a class which applies to that class and all its subclasses on which the attribute, property or setting is unset.

dynamic size
dynamic sizing
: A form of sizing wherein the image size is automatically computed at render-time.

  This only works with *automatic sizing*.

  See also:

  fixed sizing and `size`

fixed size
fixed sizing
: A form of sizing wherein the image size is set to a specific value which won't change until it is re-set.

  This works with both *manual sizing* and *automatic sizing*.

  See also:

  dynamic sizing, `set_size()`, `width` and `height`

horizontal alignment
: The horizontal position of a primary *render* output within its *padding width*.

  See also:

  Alignment

horizontal allowance
: The amount of **columns** to be left unused on the terminal screen.

manual size
manual sizing
: A form of sizing wherein **both** the width and the height are specified to set the image size.

  This form of sizing does not preserve image aspect ratio and can only be used with *fixed sizing*.

  See also:

  automatic sizing and `set_size()`

padding
: Amount of lines and columns within which to fit a primary *render* output.

  See also:

  Padding

padding height
: Amount of **lines** within which to fit a primary *render* output.

  Excess lines on either or both sides of the render output (depending on the *vertical alignment*) will be filled with spaces.

  See also:

  Padding

padding width
: Amount of **columns** within which to fit a primary *render* output.

  Excess columns on either or both sides of the render output (depending on the *horizontal alignment*) will be filled with spaces.

  See also:

  Padding

pixel ratio
: The aspect ratio with which one rendered pixel is drawn/displayed on the terminal screen.

  For *Graphics-based Render Styles*, this is ideally `1.0`.

  For *Text-based Render Styles*, this is equvalent to the *cell ratio* multiplied by 2, since there are technically two times more pixels along the vertical axis than along the horizontal axis in one character cell.

render
rendered
rendering
: The process of encoding pixel data into a byte/character **string** (possibly including escape sequences to reproduce colour and transparency).

  This string is also called the **primary** render output and **excludes** *padding*.

render method
render methods
: A unique implementation of a *render style*.

  See also:

  Render Methods

render style
render styles
style
styles
: A specific technique for rendering or displaying pixel data (including images) in a terminal emulator.

A render style (or simply *style*) is implemented by a class, often refered to as a *render style class* (or simply *style class*).

See also:

Render Styles

rendered height
The amount of **lines** that'll be occupied by a primary *render* output **when drawn (written) onto a terminal screen**.

rendered size
The amount of space (columns and lines) that'll be occupied by a primary *render* output **when drawn (written) onto a terminal screen**.

rendered width
The amount of **columns** that'll be occupied by a primary *render* output **when drawn (written) onto a terminal screen**.

scale
The fraction/ratio of an image's size that'll actually be used to *render* it.

See also:

Image scale

source
The resource from which an image instance is initialized.

See also:

`source` and `source_type`

terminal height
The amount of lines on a terminal screen at a time i.e without scrolling.

terminal size
The amount of columns and lines on a terminal screen at a time i.e without scrolling.

terminal width
The amount of columns on a terminal screen at a time.

vertical alignment
The vertical position of a primary *render* output within its *padding height*.

See also:

Alignment

vertical allowance
The amount of **lines** to be left unused on the terminal screen.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

t

term_image, 20
term_image.exceptions, 51
term_image.image, 23
term_image.utils, 53
term_image.widget, 48

INDEX

A

active terminal, **56**
alignment, **56**
allowance, **56**
alpha threshold, **56**
animated, **57**
AUTO (*term_image.image.Size attribute*), 24
auto_image_class() (*in module term_image.image*), 23
AutoCellRatio (*class in term_image*), 20
AutoImage() (*in module term_image.image*), 23
automatic size, **57**
automatic sizing, **57**
available height, **57**
available size, **57**
available width, **57**

B

BaseImage (*class in term_image.image*), 26
BlockImage (*class in term_image.image*), 36
BlockImageError, 52

C

cell ratio, **57**
clear() (*term_image.image.ITerm2Image class method*), 42
clear() (*term_image.image.KittyImage class method*), 45
clear() (*term_image.widget.UrwidImage method*), 49
clear_all() (*term_image.widget.UrwidImage static method*), 49
close() (*term_image.image.BaseImage method*), 31
close() (*term_image.image.ImageIterator method*), 47
closed (*term_image.image.BaseImage property*), 28

D

DEFAULT_QUERY_TIMEOUT (*in module term_image*), 20
descendant, **57**
disable_queries() (*in module term_image*), 21
disable_win_size_swap() (*in module term_image*), 21
draw() (*term_image.image.BaseImage method*), 31
draw_screen() (*term_image.widget.UrwidImageScreen method*), 50
DYNAMIC (*term_image.AutoCellRatio attribute*), 20
dynamic size, **57**
dynamic sizing, **57**

E

enable_queries() (*in module term_image*), 21
enable_win_size_swap() (*in module term_image*), 21

F

FILE_PATH (*term_image.image.ImageSource attribute*), 24
FIT (*term_image.image.Size attribute*), 24
FIT_TO_WIDTH (*term_image.image.Size attribute*), 24
FIXED (*term_image.AutoCellRatio attribute*), 20
fixed size, **57**
fixed sizing, **57**
flush() (*term_image.widget.UrwidImageScreen method*), 50
forced_support (*term_image.image.BaseImage property*), 28
frame_duration (*term_image.image.BaseImage property*), 28
from_file() (*in module term_image.image*), 23
from_file() (*term_image.image.BaseImage class method*), 32
from_url() (*in module term_image.image*), 23
from_url() (*term_image.image.BaseImage class method*), 33

G

get_available_raw_input() (*term_image.widget.UrwidImageScreen method*), 51
get_cell_ratio() (*in module term_image*), 22
get_terminal_name_version() (*in module term_image.utils*), 53
get_terminal_size() (*in module term_image.utils*), 53
GraphicsImage (*class in term_image.image*), 36
GraphicsImageError, 52

H

height (*term_image.image.BaseImage property*), 28
horizontal alignment, **57**
horizontal allowance, **57**

I

image (*term_image.widget.UrwidImage property*), 49
ImageIterator (*class in term_image.image*), 46
ImageMeta (*class in term_image.image*), 25
ImageSource (*class in term_image.image*), 24
InvalidSizeError, 52
is_animated (*term_image.image.BaseImage property*), 29
is_supported (*term_image.AutoCellRatio attribute*), 20
is_supported() (*term_image.image.BaseImage class method*), 33
ITerm2Image (*class in term_image.image*), 37
ITerm2ImageError, 52

J

jpeg_quality (*term_image.image.ITerm2Image property*), 40

K

KittyImage (*class in term_image.image*), 42
KittyImageError, 52

L

lock_tty() (*in module term_image.utils*), 53
loop_no (*term_image.image.ImageIterator property*), 47

M

manual size, **58**
manual sizing, **58**
module
    term_image, 20
    term_image.exceptions, 51
    term_image.image, 23
    term_image.utils, 53
    term_image.widget, 48

N

n_frames (*term_image.image.BaseImage property*), 29
native_anim_max_bytes (*term_image.image.ITerm2Image property*), 41

O

ORIGINAL (*term_image.image.Size attribute*), 24
original_size (*term_image.image.BaseImage property*), 29

P

padding, **58**
padding height, **58**
padding width, **58**
PIL_IMAGE (*term_image.image.ImageSource attribute*), 24
pixel ratio, **58**

R

read_from_file (*term_image.image.ITerm2Image property*), 41
read_tty_all() (*in module term_image.utils*), 54
render, **58**
render method, **58**
render methods, **58**
render style, **58**
render styles, **58**
rendered, **58**
rendered height, **59**
rendered size, **59**
rendered width, **59**
rendered_height (*term_image.image.BaseImage property*), 29
rendered_size (*term_image.image.BaseImage property*), 29
rendered_width (*term_image.image.BaseImage property*), 29
rendering, **58**

S

scale, **59**
scale (*term_image.image.BaseImage property*), 29
scale_x (*term_image.image.BaseImage property*), 30
scale_y (*term_image.image.BaseImage property*), 30
seek() (*term_image.image.BaseImage method*), 34
seek() (*term_image.image.ImageIterator method*), 47
set_cell_ratio() (*in module term_image*), 22
set_error_placeholder() (*term_image.widget.UrwidImage class method*), 49
set_query_timeout() (*in module term_image*), 22
set_render_method() (*term_image.image.BaseImage class method*), 34
set_size() (*term_image.image.BaseImage method*), 34
Size (*class in term_image.image*), 24
size (*term_image.image.BaseImage property*), 30
source, **59**
source (*term_image.image.BaseImage property*), 30
source_type (*term_image.image.BaseImage property*), 30
style, **58**
style (*term_image.image.ImageMeta property*), 25
StyleError, 52
styles, **58**

T

tell() (*term_image.image.BaseImage method*), 35
term_image
    module, 20
term_image.exceptions
    module, 51
term_image.image
    module, 23
term_image.utils
    module, 53
term_image.widget
    module, 48
TermImageError, 51
TermImageWarning, 51
terminal height, **59**
terminal size, **59**
terminal width, **59**
TextImage (*class in term_image.image*), 36
TextImageError, 52

U

URL (*term_image.image.ImageSource attribute*), 24
URLNotFoundError, 51
UrwidImage (*class in term_image.widget*), 48
UrwidImageCanvas (*class in term_image.widget*), 49
UrwidImageError, 52
UrwidImageScreen (*class in term_image.widget*), 50

V

vertical alignment, **59**
vertical allowance, **59**

W

width (*term_image.image.BaseImage property*), 30
write() (*term_image.widget.UrwidImageScreen method*), 51
write_tty() (*in module term_image.utils*), 54