# The latex-lab-graphic package
# Tagging of included graphics

LaTeX Project*

v0.80j 2026-09-15

**Abstract**

The following code implements a first draft for the tagging of graphics included with `\includegraphics` and the `picture` environment.

## 1 Introduction

Tagging of pictures is non trivial.

1. Pictures generally can have various purposes and so different *tagging modes* must be provided:
   - The pictures can be purely *ornamental* and *decorative*, e.g., (always/usually) some page border. This should normally be tagged as *artifact*, either in a `artifact` MC-chunk or as an `Artifact` structure[1].
   - The pictures can be *illustrative figures*. This should normally be tagged as a `Figure` structure with *alternative text*.
   - The pictures can represent a symbol. This should be tagged as a `Span` structure element with an `/ActualText` mapping to some Unicode Codepoint(s) (or perhaps even directly in the stream with a Span BDC with an `/ActualText`).
   - The pictures can also be intended for consumption as normal *text* For example the todonotes package uses a ti*k*Z picture to surround the text in a node with some colored frame and background. In this case the text (which can contain more elements like lists) should be tagged as usual and put in an `Aside` structure while the decorative elements should be marked up as artifacts.
   - There are also case where no tagging is wanted, e.g. because the tagging is done by surrounding commands, in this case the begin/end sockets should be transparent and do nothing.
   - And naturally there can be more complicated scenarios with mixtures of these elements, e.g., some text in nodes in a ti*k*Z picture can be meaningful while other nodes still should be tagged as artifact.

*Initial implementation done by Ulrike Fischer

[1] The second option exists only in PDF 2.0

2. The various packages that allow to draw pictures uses lots of boxes and moves them around and that makes is not easy to get the tagging right – especially with pdflatex where one has to insert the literals at the right time.

3. When the picture is tagged as `Figure`, the PDF-UA standards require an attribute with a `BBox` key in the structure. That BBox describes the placement of the picture on the page, and this typically require some low-level hacking into the picture code to calculate the values.

The code here handles directly only the tagging of pictures included with `\includegraphics` and the `picture` environment. But the used method and the documentation tries to stay as general as possible, to make it easy to adapt the code to pictures drawn with other packages like l3draw, tikz, pstricks, luamplibs or similar packages.

# 2 General implementation needs and ideas

## 2.1 User interfaces

The tagging of pictures can not be done fully automatically: user have to add alternative text or mark up a picture as an artifact. It is important that the relevant user interfaces are similar across all picture/graphic packages to make it easy for authors to adapt their documents.

We therefore recommend package authors to implement an interface a key-value interface like the following.

### 2.1.1 Tagging mode of individual graphics

To change the tagging mode of an individual graphic, the keys `artifact` (decorations), `alt` (illustrative), and `actualtext` (symbol) should be used. The keys `alt` (and also `actualtext`) take a text as argument.

```
\includegraphics[artifact]{example-image}
\begin{tikzpicture}[artifact] ... \end{tikzpicture}
\tikz[artifact]{...}
\begin{picture}[artifact]...\end{picture}

\includegraphics[alt=example image]{example-image}
\begin{tikzpicture}[alt=\myalttext] ... \end{tikzpicture}
\tikz[alt=example image]{...}
\begin{picture}[alt=example image]...\end{picture}

\includegraphics[actualtext=A]{example-image-A}
\begin{tikzpicture}[actualtext=A] ... \end{tikzpicture}
\tikz[actualtext=A]{...}
\begin{picture}[actualtext=A]...\end{picture}
```

Additionally, a key `tagging-setup` can be provided that allows to handle more complicated cases, e.g.,

```
\includegraphics[tagging-setup={off}]{example-image} %no tagging at all
\begin{tikzpicture}[tagging-setup={artifact}] ... \end{tikzpicture}
\begin{tikzpicture}[tagging-setup={text}] ... \end{tikzpicture}
\begin{tikzpicture}[tagging-setup={alt=Water (H20),tag=Formula}] ... \end{tikzpicture}
```

This key is fully implemented for `\includegraphics` and `picture` but only partially in the ti*k*Z module.

### 2.1.2 Setting the tagging mode for a scope

Packages can also provide interfaces to change the tagging mode for all pictures or nodes in a scope.

If this is done side-effects on the tagging of other picture types must be considered: If the generic sockets declared below are used (which are then used also by `\includegraphics`, `picture` and perhaps more environments) a change of the tagging mode can affect all pictures types using these sockets in the same scope.

This means that packages that want to offer "scope support" but restrict it to their own picture type should either use their own sockets modelled after the generic sockets, or use some specific variable to control the change of the tagging mode.

With ti*k*Z "scope support" could be implemented rather easily as it has a setup command anyway. So with the implementation in latex-lab-tikz, all these work as expected

```
\tikzset{artifact}
\tikzset{alt=a text}
\tikzset{actualtext=A}
\tikzset{tagging-setup=text}
```

With `\includegraphics` the standard `\setkeys` can be used:

```
\setkeys{Gin}{alt=a text}
\setkeys{Gin}{artifact}
\setkeys{Gin}{actualtext=A}
```

This will then also affect `picture` environments in the scope.

## 2.2 Default tagging mode

If none of the keys are used, the picture code must chose a default tagging mode. The best one depends on the graphic type and should be chosen as needed and then documented.

With `\includegraphics` we use as default the illustrative mode, tag it as `Figure` structure, use the file name as alternative text and issue a warning that a real alternative text is missing.

With the `picture` environment we use as default the illustrative mode to, but use a fix text as alternative text and issue a warning too.

With ti*k*Z (a first implementation is currently in latex-lab-tikz), the default is the text mode, which means that a screen reader will read the text in the nodes in the order they appear in the code. With this default, e.g. `\todo`'s from the `todonotes` are correctly tagged.

In this implementation the `artifact` key can also be applied on nodes and will remove them from the tagging.

```
\begin{tikzpicture}
\node[draw=red](x){Important!};
\node[artifact,fill=blue,anchor=west] at (x.east){\phantom{Ip}};
\end{tikzpicture}
```

### 2.3 Sockets, plugs and commands

**sockets** The example implementations make use of up-to five *tagging sockets*. The sockets are all used by the various graphic codes inside a group:

- An initialization (`init`) socket that handles the key-val argument and setups the tagging mode by switching the plugs of the other tagging sockets.
- Two sockets that are used at the `begin` and `end` of the picture and setup the main structure element. The `end` socket typically also executes if needed the code to calculate the BBox attribute and add it to the structure. After the `begin` socket tagging is suspended, and before the `end` socket resumed.
- Two sockets for the text mode that are used around places where a picture contains text. Tagging must be resumed before this sockets if they should do anything at all.

The sockets take arguments that allows some configuration (e.g. to use a special command for the BBox calculation) and the plugs are coded so that they can be used in more than one picture type (they are shared here between `\includegraphics` and `picture`) but there is no obligation to use them in every graphic code. A package can declare its own sockets and plugs. See 2.1.2 for some discussion why this can be a good idea.

**plugs** The main `begin` and `end` sockets should normally have four *plugs* for the different modes: `alt`, `actualtext`, `artifact` and `text`. To disable tagging, `\SuspendTagging` can be used, alternatively for sockets with zero or one argument, the predefined `noop` plug can be assigned, and for sockets with two arguments (if the socket has been declared with `\NewTaggingSocket` the `transparent` socket which lets pass through the second argument.

The initialization socket and the texts sockets typically have only one additional *plug* that is used when tagging is active.

**commands** A command to store the position of a reference point on the page and a command to calculate the BBox from this reference point and the size of the picture are needed. The second command must also add the attribute to the main structure element (how to do this can be seen in the implementation). These commands are typically specific for a picture type.

We now describe how this general principles have be implemented in the concrete examples of the `\includegraphics` command, the `picture` environment, TODO! and a simple environment for l3draw commands.

## 3 Sockets, plugs, commands

The code defines the following generic sockets and commands.

- `graphic/init` with the plug `default`
- `graphic/begin` (one argument) with the plugs `alt`, `actualtext`, `artifact`, `text`, `off`. The argument allows to set a default alternative text.
- `graphic/end` (two arguments) with the plugs `alt`, `actualtext`, `artifact`, `text`, `off`. The first argument of the socket typically receives the command to calculate the BBox. This command often contains a savepos command and so has to go before the graphic box. The second argument allows to insert the box between this command and the tagging commands.
- `graphic/text/begin` no argument, with the plug `default`
- `graphic/text/end` no argument with the plug `default`

`\l_tag_graphic_mode_tl` This variable holds the current active graphic mode. It is e.g. used to test if text should resume tagging.

# 4 Make `\includegraphics` tagging aware

Tagging of graphics included with `\includegraphics` is at a first glance easier to handle than, e.g., a ti*k*Z graphic, as there is only a simple box with a picture and no text content to consider. One would think that adding some structure commands around a box shouldn't pose much problems.

But things are actually not so easy.

At first such graphics are inserted into the PDF as XObjects and there are two ways to add an such an XObject to a structure: similar to text as a marked content item (by surrounding it with `\tagmcbegin` and `\tagmcend`) or by referencing the XObject with an OBJR object (similar to a link annotation). Which method is more sensible (and if it actually matters) is unknown. Currently the first method is used as the second would changes in the backend files.

At second—and this is actually a *much* bigger problem[2]—if the graphic is tagged as an illustrative picture the `Figure` structure element should have an attribute with an `BBox` entry. The value of this `BBox` is an array of four numbers that gives the coordinates of the left, bottom, right, and top edges of the structure element's bounding box on the page. That is the rectangle that completely encloses its *visible* content and so has not necessarily the same size as the TeX bounding box: if `viewport` or `trim` is used and the graphic is not clipped, the visible content can be larger. It turned out to be extremely tricky to get a sensible result, and there are still open problems and restrictions.

## 4.1 The BBox calculation

The reference point on the page is retrieved with `\tex_savepos:D` and a property just before the graphic.

Getting from this the `BBox` can be quite straightforward for a graphic that is used once as is. But graphics can be trimmed, scaled, reflected, rotated and reused in various ways. These transformations typically involve a mix of TEX commands that shift a

[2]Which shows also in the amount of code dedicated here only to this problem.

box or change the bounding box and backend commands that insert a pdfliteral with a transformation matrix. Calculating the correct `BBox` in all cases is not possible without rewriting large parts of the graphics and graphicx packages. Problematic are

- manipulations through external box commands (`\rotatebox`, `\reflectbox`, `\scalebox`). Their implementation do not pass the transformation matrix in a way that allows to track the changes for the `BBox` of an included graphic: sometimes the values are set to late (after the box is already stored), and often the values are not grouped and can leak out from earlier uses of the commands.
- some combination of keys in the optional argument of `\includegraphics`. Examples are `origin` and multiple calls to `scale` and `angle`) as they internally call the box commands. Examples of failing combinations can be found in the test file `graphic-faults`.
- graphics that are stored in a box and reused: to get the `BBox` one has to set a label that stores the position with `\pdfsavepos`, and if a box is reused one gets multiply defined labels. One possible solution here is to make use of the new delayed `\pdfliteral`. It allows to change the label names in the shipout, but this requires careful tracking the box usages and so various kernel changes.

Therefore a correct BBox is currently implemented only for simple `\includegraphics` and the keys `viewport`, `trim`, `scale` and `angle` (used at most once).

Currently not supported are

- graphics inside `\rotatebox`, `\reflectbox`, `\scalebox`.

  TODO: A new implementation with `l3graphics` and `l3box` is probably needed here.
- multiple uses of the `scale` and `angle` keys
- multiple use of graphics stored in boxes. For such graphics automated tagging should be probably deactivated when storing the content and tagging should be added around the `\usebox`. (How to proceed when content is saved in boxes needs generally more testing).

## 4.2 User interface

As suggested above the code (re)defines keys for `\includegraphics` to add the recommended interfaces `alt`, `artifact` and `actualtext`. It also offers some keys specific to `\includegraphics`:

**`alt`** This key is already defined in the graphicx package but redefined here to switch to the illustrative mode and to add its value as alternative text.

**`actualtext`** This switches to the actualtext mode. This is useful for small graphics that represent single chars or a short word like a logo. If `actualtext` is used, the graphics is not enclosed in `Figure` structure but in a `Span` structure and no `/BBox` attribute is added. [3]

**`artifact`** This tags the graphic as an artifact.

[3] This in accordance with (the draft of) PDF/UA-2 but violates perhaps PDF/UA-1.

`tagging-setup` This key takes as argument a key-list, which can contain the following keys:

- `alt=`⟨*text*⟩ This a second way to tag the graphic as figure with alternative text. `alt` without value will tag as figure but not change the text variable, `alt=` will empty the text variable (and typically trigger a warning).
- `actualtext=`⟨*text*⟩ This a second way to tag the graphic as a symbol with actualtext.
- `artifact` This a second way to tag the graphic as artifact.
- `text` This switches to text mode.
- `off` When used tagging will be stopped completely. It is then the responsibility of the surrounding code to add appropriate tagging commands.
- `tag=`⟨*name*⟩ This switches to the illustrative mode but uses ⟨*name*⟩ as tag name in the structure instead of the default `Figure`. This can for example be used to tag an image of a formula with `Formula`.

`adjust-BBox` If the calculated `/BBox` values are wrong they can be corrected with this key. It expects four dimensions that are added to the `/BBox` values.

The code also add to the `debug` key of `\DocumentMetadata` the value `BBox`. This adds a half transparent red layer showing the calculated BBox.

```
\DocumentMetadata{tagging=on,debug=BBox}
```

## 4.3 Hooks

The three key definitions `alt`, `actualtext` and `artifact` used by `\includegraphics` contain hooks, named `Gin/alt`, `Gin/actualtext`, and `Gin/artifact`, respectively. The first two take two arguments: the (with `\pdf_purify:nN`) purified value of the key that is also used in the PDF and the raw value of the key. The hooks are processed even if tagging is not activated. With them, it is for example possible to store the alternative text:

```
\AddToHookWithArguments{Gin/alt}{\gdef\myalttext{#2}}
\includegraphics[alt=Hello World]{example-image-A}
The alt text of the graphic was \myalttext.
```

Please note

- The hooks are also executed if the keys are set with `\setkeys{Gin}`.
- The hooks are not executed in the picture environment or in tikz pictures as they use different keys.
- The hooks are obviously not executed if the keys are not used. That means they can't be used directly to issue a warning or an error that an alt text is missing. For this some additional logic is needed, e.g., a counter in the `cmd/includegraphics/before` hook, that allows to identify the graphics and to store the alt or actual text individually.

# 5 Make picture tagging aware

## 5.1 User interface

The original picture environment doesn't have a optional argument with key-value value processing. So the code changes that and then provides the keys already discussed for \includegraphics.

## 5.2 BBox calculation

The BBox calculation is much simpler than for \includegraphics as the code simply takes the declared size from the picture arguments.

## 5.3 Tagging in text mode

A picture can contain \put commands with text content, tagging this in text mode could make sense in some cases. But it is not quite clear if one can/should redefine the \put command. If text tagging is wanted we suggest to define a dedicated command along these lines:

```
\NewDocumentCommand\picturenode{O{}r()m}
  {
   \group_begin:
   \keys_set:nn{tag/graphic}{#1}
   \str_if_eq:VnT\l_tag_graphic_mode_tl {text}
    {\tag_resume:n{\picturenode}}
   \tag_socket_use:n{graphic/text/begin}
   \put(#2){#3}
   \tag_socket_use:n{graphic/text/end}
   \group_end:
  }
```

# 6 Make an l3draw environment tagging aware

The newest l3draw version (in the latex3 github) has all needed data to define a command to retrieve the BBox and to build a tagging aware environment. l3draw currently has no dedicated function to add text, instead one has to store the text in a box and then reuse it, so similar to the picture environment a dedicated command with tagging awareness is suggested.

A command to calculate the BBox can for example be defined like this

```
\cs_new:Npn\draw_tag_bbox_attribute:
 {
   \tl_set:Ne \g__tag_graphic_lx_tl
     {
       \dim_to_decimal_in_bp:n
         {
           \property_ref:een {draw.\int_use:N\g_draw_id_int}{xpos}{0}sp
           +
           \g_draw_bb_xmin_dim
```

```
      }
  }
\tl_set:Ne \g__tag_graphic_ly_tl
  {
    \dim_to_decimal_in_bp:n
      {
        \property_ref:een {draw.\int_use:N\g_draw_id_int}{ypos}{0}sp
        +
        \g_draw_bb_ymin_dim
      }
  }
\tl_set:Ne \g__tag_graphic_ux_tl
  {
    \dim_to_decimal_in_bp:n
      {
        \property_ref:een {draw.\int_use:N\g_draw_id_int}{xpos}{0}sp
        +
        \g_draw_bb_xmax_dim
      }
  }
\tl_set:Ne \g__tag_graphic_uy_tl
  {
    \dim_to_decimal_in_bp:n
      {
        \property_ref:een {draw.\int_use:N\g_draw_id_int}{ypos}{0}sp
        +
        \g_draw_bb_ymax_dim
      }
  }
\bool_if:NT\l__tag_graphic_debug_bool
 {
   \__tag_graphic_show_bbox:VVVVne
    \g__tag_graphic_lx_tl
    \g__tag_graphic_ly_tl
    \g__tag_graphic_ux_tl
    \g__tag_graphic_uy_tl
    {red}
    {draw.\int_use:N\g_draw_id_int}
 }
\tag_struct_gput:ene
    {\tag_get:n{struct_num}}
    {attribute}
    {
      /O /Layout /BBox~
      [
        \dim_to_decimal_in_bp:n
         {
          \property_ref:een {draw.\int_use:N\g_draw_id_int}{xpos}{0}sp
           +
          \g_draw_bb_xmin_dim
```

```
            }
           \c_space_tl
           \dim_to_decimal_in_bp:n
            {
              \property_ref:een {draw.\int_use:N\g_draw_id_int}{ypos}{0}sp
              +
              \g_draw_bb_ymin_dim
            }
           \c_space_tl
           \dim_to_decimal_in_bp:n
            {
              \property_ref:een {draw.\int_use:N\g_draw_id_int}{xpos}{0}sp
              +
              \g_draw_bb_xmax_dim
            }
           \c_space_tl
           \dim_to_decimal_in_bp:n
            {
              \property_ref:een {draw.\int_use:N\g_draw_id_int}{ypos}{0}sp
              +
              \g_draw_bb_ymax_dim
            }
        ]
      }
  }
```

A tagging aware environment can then be defined like this

```
\NewDocumentEnvironment{tagged-draw}{O{}}
 {\leavevmode
  \tag_socket_use:nn{graphic/init}{#1}
  \tag_socket_use:nn{graphic/begin}{tagged-draw~environment}
  \tag_suspend:n{\draw} %
  \draw_begin:\ignorespaces
 }
 {
  \draw_end:
  \tag_resume:n{\draw}
  \tag_socket_use:nnn{graphic/end}{\draw_tag_bbox_attribute:}{}
 }
```

And a command for tagged text nodes could look like this

```
\cs_new_protected:Npn \draw_text_node:nnn #1#2#3 %#1 keyval, #2 point, #3 text
 {
   \group_begin:
   \keys_set:nn{tag/graphic}{#1}
   \str_if_eq:VnT\l_tag_graphic_mode_tl {text}
      {\tag_resume:n{\draw_node:nn}}
   \tag_socket_use:n{graphic/text/begin}
   \hbox_set:Nn \l_tmpa_box{#3}
```

```
  \draw_box_use:Nn\l_tmpa_box {#2}
  \tag_socket_use:n{graphic/text/end}
  \group_end:
}
```

```
⟨@@=tag⟩
⟨*package⟩
```

# 7 Implementation

```
\ProvidesExplPackage {latex-lab-testphase-graphic} {\ltlabgraphicdate} {\ltlabgraphicversion}
  {Code related to the tagging of graphics}
```

a variant

```
\cs_generate_variant:Nn \tag_socket_use:nn {ne}
```

__tag_graphic_savepos:n this is the command which stores the position. Similar to zref-savepos it uses two savepos commands for the case that bidi changes the processing order.

```
\cs_new_protected:Npn\__tag_graphic_savepos:n #1
 {
  \tex_savepos:D
  \property_record:nn{#1}{xpos,ypos,abspage}
  \tex_savepos:D
 }
\cs_generate_variant:Nn \__tag_graphic_savepos:n {e}
```

(*End of definition for* `\__tag_graphic_savepos:n`.)

## 7.1 Variables

\l__tag_graphic_alt_tl
\l__tag_graphic_actual_tl
\l_tag_graphic_mode_tl

These variables are related to the tagging. Variables for the alt text, the actualtext and the structure tag. The variable that holds the tagging mode is public, so that commands can test for it, and e.g. restart tagging in text mode.

```
\tl_new:N \l__tag_graphic_alt_tl
\tl_new:N \l__tag_graphic_actual_tl
```

(*End of definition for* `\l__tag_graphic_alt_tl`, `\l__tag_graphic_actual_tl`, *and* `\l_tag_graphic_mode_tl`. *This variable is documented on page* 5.)

\l__tag_graphic_debug_bool A boolean for debug code

```
\bool_new:N \l__tag_graphic_debug_bool
```

(*End of definition for* `\l__tag_graphic_debug_bool`.)

The rest of the variables are related to the BBox calculation in `\includegraphics`.

\g__tag_graphic_int This is used to get unique labels in the savepos code.

```
\int_new:N\g__tag_graphic_int
```

(*End of definition for* `\g__tag_graphic_int`.)

\g__tag_graphic_lx_tl
\g__tag_graphic_ly_tl
\g__tag_graphic_ux_tl
\g__tag_graphic_uy_tl
bboxcorr_seq \l__tag_graphic_bboxcorr_bool

This commands will hold the calculated BBox values. Local variables would probably work too, but global variables can be easier retrieved in tests and debugging code ...

```
\tl_new:N \g__tag_graphic_lx_tl
\tl_new:N \g__tag_graphic_ly_tl
\tl_new:N \g__tag_graphic_ux_tl
\tl_new:N \g__tag_graphic_uy_tl
\seq_new:N\l__tag_graphic_bboxcorr_seq
\bool_new:N\l__tag_graphic_bboxcorr_bool
```

(*End of definition for* `\g__tag_graphic_lx_tl` *and others.*)

\l__tag_graphic_currentlabel_tl

This holds the label name of the savepos.

```
\tl_new:N \l__tag_graphic_currentlabel_tl
```

(*End of definition for* `\l__tag_graphic_currentlabel_tl`.)

\l__tag_graphic_sin_fp
\l__tag_graphic_cos_fp
\l__tag_graphic_scale_fp
\l__tag_graphic_lxly_fp
\l__tag_graphic_lxuy_fp
\l__tag_graphic_uxly_fp
\l__tag_graphic_uxuy_fp
\l__tag_graphic_ux_fp
\l__tag_graphic_ly_fp
\l__tag_graphic_lx_fp
\l__tag_graphic_uy_fp
\l__tag_graphic_trim_ux_fp
\l__tag_graphic_trim_ly_fp
\l__tag_graphic_trim_lx_fp
\l__tag_graphic_trim_uy_fp

A bunch of fp-variables (we don't use tl-vars, to avoid to have to take care about minus signs everywhere)

```
\fp_new:N\l__tag_graphic_sin_fp
\fp_new:N\l__tag_graphic_cos_fp
\fp_new:N\l__tag_graphic_lxly_fp
\fp_new:N\l__tag_graphic_lxuy_fp
\fp_new:N\l__tag_graphic_uxly_fp
\fp_new:N\l__tag_graphic_uxuy_fp
\fp_new:N\l__tag_graphic_ux_fp
\fp_new:N\l__tag_graphic_ly_fp
\fp_new:N\l__tag_graphic_lx_fp
\fp_new:N\l__tag_graphic_uy_fp
```

this holds the scale value. Either `\Gin@scalex` or (if that is !) `\Gin@scaley`

```
\fp_new:N\l__tag_graphic_scale_fp
```

the follow variables hold the four trim values (or the equivalent calculated values if viewport is used.

```
\fp_new:N\l__tag_graphic_trim_ux_fp
\fp_new:N\l__tag_graphic_trim_ly_fp
\fp_new:N\l__tag_graphic_trim_lx_fp
\fp_new:N\l__tag_graphic_trim_uy_fp
```

(*End of definition for* `\l__tag_graphic_sin_fp` *and others.*)

## 7.2 Tagging sockets

The sockets can perhaps not be shared between `\includegraphics` and `picture` but for now we try to do it, with the exception of the init socket.

The begin socket takes an argument to allow to pass some configuration. The end socket takes two argument to allow to calculate the BBox before outputting the box.

agsupport/graphic/init (*socket*)
gsupport/graphic/begin (*socket*)
tagsupport/graphic/end (*socket*)
ort/graphic/text/begin (*socket*)
raphic/graphic/text/end (*socket*)

```
\NewTaggingSocket{graphic/init}{1}
\NewTaggingSocket{graphic/begin}{1}
\NewTaggingSocket{graphic/end}{2}
\NewTaggingSocket{graphic/text/begin}{0}
\NewTaggingSocket{graphic/text/end}{0}
```

## 7.3 Tagging plugs

### 7.3.1 Initialization of the tagging mode

tagsupport/graphic/init) (*plug*)
tagsupport/graphic/init) (*plug*)

```
\NewTaggingSocketPlug{graphic/init}{default}
  {
   \keys_set:nn{tag/graphic}{#1}
   \ExpandArgs{no}\AssignTaggingSocketPlug{graphic/begin}{\l_tag_graphic_mode_tl}
   \ExpandArgs{no}\AssignTaggingSocketPlug{graphic/end}{\l_tag_graphic_mode_tl}
  }
\AssignTaggingSocketPlug{graphic/init}{default}
```

### 7.3.2 Main begin and end sockets

agsupport/graphic/begin) (*plug*)
(tagsupport/graphic/end) (*plug*)

These plugs handle the graphic as a figure. Around the graphic is a `Figure` environment which will use an alt text given in the optional argument and internally tagging is suspended. The Bbox will be set (after the second compilation) to the size of the declared size.

```
\msg_new:nnn { tag } { alt-text-missing }
 {
   Alternative~text~for~graphic~is~missing.\\
   Using~'#1'~instead.
 }
\NewTaggingSocketPlug{graphic/begin}{alt}
  {
    \tag_mc_end_push:
    \tl_if_empty:NT\l__tag_graphic_alt_tl
      {
        \msg_warning:nne{tag}{alt-text-missing}{#1}
        \pdf_purify:nN {#1}\l__tag_graphic_alt_tl
      }
    \tag_struct_begin:n
      {
        tag=\UseStructureName{graphic},
        alt=\l__tag_graphic_alt_tl,
      }
    \tag_mc_begin:n{}
  }
\NewTaggingSocketPlug{graphic/end}{alt}
  {
     #1
     #2
     \tag_mc_end:
     \tag_struct_end:
     \tag_mc_begin_pop:n{}
  }
```

agsupport/graphic/begin) (*plug*)
(tagsupport/graphic/end) (*plug*)

This plug handles the picture as a symbol with an actualtext. It tags the content as a Span and expects an actualtext. Internally tagging is suspended.

```
\NewTaggingSocketPlug{graphic/begin}{actualtext}
  {
   \tag_mc_end_push:
   \tag_struct_begin:n{tag=\UseStructureName{graphic/symbol},actualtext=\l__tag_graphic_actua
```

```
    \tag_mc_begin:n{}
  }
\NewTaggingSocketPlug{graphic/end}{actualtext}
  {
   #2
   \tag_mc_end:
   \tag_struct_end:
   \tag_mc_begin_pop:n{}
  }
```

agsupport/graphic/begin) (*plug*)
(tagsupport/graphic/end) (*plug*)

This plug handles the picture as an artifact, as decoration. So it is surrounded by an artifact MC and internal text does not restart tagging.

```
\NewTaggingSocketPlug{graphic/begin}{artifact}
  {
    \tag_mc_end_push:
    \tag_mc_begin:n{artifact}
  }
\NewTaggingSocketPlug{graphic/end}{artifact}
  {
    #2
    \tag_mc_end:
    \tag_mc_begin_pop:n{}
  }
```

agsupport/graphic/begin) (*plug*)
(tagsupport/graphic/end) (*plug*)

This plug can be used for text tagging. It basically does the same as the artifact plug. The main reason that it exists is that it looks more consistent and that we can test for the plug name and so restart tagging in places where this is wanted. (tagging can not be resumed inside a tagging hook, so has to use some external method).

```
\NewTaggingSocketPlug{graphic/begin}{text}
  {
    \tag_mc_end_push:
    \tag_mc_begin:n{artifact}
  }
\NewTaggingSocketPlug{graphic/end}{text}
  {
    #2
    \tag_mc_end:
    \tag_mc_begin_pop:n{}
  }
```

agsupport/graphic/begin) (*plug*)
(tagsupport/graphic/end) (*plug*)

This plug can be used for text tagging. It basically does the same as the artifact plug. The main reason that it exists is that it looks more consistent and that we can test for the plug name and so restart tagging in places where this is wanted. (tagging can not be resumed inside a tagging hook, so has to use some external method).

```
\NewTaggingSocketPlug{graphic/begin}{off}{}
\NewTaggingSocketPlug{graphic/end}{off}{#2}
```

By default we use the alt plugs

```
\AssignTaggingSocketPlug{graphic/begin}{alt}
\AssignTaggingSocketPlug{graphic/end}{alt}
```

port/graphic/text/begin) (*plug*) These sockets are used inside the text plugs and ends the previous mc and restarts it after
upport/picture/text/end) (*plug*) the text. Not used by `\includegraphics`. Unclear if they can be used by the `picture` environment.

```
\NewTaggingSocketPlug{graphic/text/begin}{default}
  {
   \tag_mc_end:
   \tag_mc_begin:n{}
  }
\NewTaggingSocketPlug{graphic/text/end}{default}
  {
   \tag_mc_end:
   \tag_mc_begin:n{artifact}
  }
\AssignTaggingSocketPlug{graphic/text/begin}{default}
\AssignTaggingSocketPlug{graphic/text/end}{default}
```

## 7.4 Generic keys

The keys are used by picture and `\includegraphics` (through the `tagging-setup` key

```
\keys_define:nn{tag/graphic}
   {
     ,mode .code:n =
        {
          \tl_set:Ne\l_tag_graphic_mode_tl{#1}
        }
     ,alt .code:n =
        {
          \pdf_purify:nN {#1}\l__tag_graphic_alt_tl
          \tl_set:Nn\l_tag_graphic_mode_tl{alt}
        }
     ,artifact .code:n =
        {
          \tl_set:Nn\l_tag_graphic_mode_tl{artifact}
        }
     ,actualtext .code:n =
        {
          \pdf_purify:nN {#1}\l__tag_graphic_actual_tl
          \tl_set:Nn\l_tag_graphic_mode_tl{actualtext}
        }
     ,text .code:n =
        {
          \tl_set:Nn\l_tag_graphic_mode_tl{text}
        }
     ,off .code:n =
        {
          \tl_set:Nn\l_tag_graphic_mode_tl{off}
        }
     ,adjust-BBox .code:n =
      {
        \bool_set_true:N \l__tag_graphic_bboxcorr_bool
        \seq_set_split:Nnn\l__tag_graphic_bboxcorr_seq{~}{#1~0pt~0pt~0pt~0pt}
      }
     ,
```

```
    tagging-setup .code:n=
     {
       \keys_set:nn { tag/graphic }{#1}
     }
```

only for legacy. Should perhaps be removed?

```
   ,tag .code:n =
     {
       \str_case:nnF {#1}
         {
           {artifact}
           {
             \tl_set:Nn\l_tag_graphic_mode_tl{artifact}
           }
           {false}{\tag_suspend:n{picture}}
         }
         {
           \AssignStructureRole{graphic}{#1}
           \tl_set:Nn\l_tag_graphic_mode_tl{alt}
         }
     }
  }
```

## 7.5 Tagging support for \includegraphics

### 7.5.1 User interface: Additional keys.

We also ensure that graphicx is loaded for the keyval support. At first a command to hold the tagging mode.

```
\tl_new:N  \l_tag_graphic_mode_tl
\tl_set:Nn \l_tag_graphic_mode_tl {alt} %TODO think about the right default.

\hook_new_with_args:nn {Gin/alt} {2}
\hook_new:n {Gin/artifact}
\hook_new_with_args:nn {Gin/actualtext}{2}
\AddToHook{package/graphicx/after}[latex-lab]
  {
    \define@key{Gin}{alt}
      {
         \pdf_purify:nN {#1}\l__tag_graphic_alt_tl
         \exp_args:Nnno\hook_use:nnw {Gin/alt}{2}{\l__tag_graphic_alt_tl}{#1}
         \tl_set:Nn\l_tag_graphic_mode_tl {alt}
      }
    \define@key{Gin}{artifact}[]
      {
        \hook_use:n{Gin/artifact}
        \tl_set:Nn\l_tag_graphic_mode_tl{artifact}
      }
    \define@key{Gin}{actualtext}
     {
       \pdf_purify:nN {#1}\l__tag_graphic_actual_tl
       \exp_args:Nnno\hook_use:nnw {Gin/actualtext}{1}{\l__tag_graphic_actual_tl}{#1}
       \tl_set:Nn\l_tag_graphic_mode_tl{actualtext}
     }
```

```
    \define@key{Gin}{tagging-setup}
     {
       \keys_set:nn { tag/graphic}{#1}
     }
    \define@key{Gin}{adjust-BBox}
      {
        \bool_set_true:N \l__tag_graphic_bboxcorr_bool
        \seq_set_split:Nnn\l__tag_graphic_bboxcorr_seq{~}{#1~0pt~0pt~0pt~0pt}
      }
```

only for legacy, no longer documented

```
    \define@key{Gin}{tag}
      {
        \str_case:nnF {#1}
          {
            {artifact}
            {
              \tl_set:Nn\l_tag_graphic_mode_tl{artifact}
            }
            {false}{\tag_suspend:n{Gin}}
          }
          {
            \AssignStructureRole{graphic}{#1}
            \tl_set:Nn\l_tag_graphic_mode_tl{alt}
          }
      }
  }
\AddToHook{package/graphics/after}[latex-lab]
  {\RequirePackage{graphicx}}
```

### 7.5.2 Patching graphics commands

All changes are currently done in `\Gin@setfile`. We mainly have to add the sockets in the right place and to rearrange a bit the `\ifGin@draft` test.

```
\AddToHook{package/graphics/after}
 {
  \def\Gin@setfile#1#2#3{%
  \ifx\\#2\\\Gread@false\fi
  \ifGin@bbox\else
    \ifGread@
      \csname Gread@%
         \expandafter\ifx\csname Gread@#1\endcsname\relax
           eps%
         \else
           #1%
         \fi
      \endcsname{\Gin@base#2}%
    \else
      \Gin@nosize{#3}%
    \fi
  \fi
  \Gin@viewport@code
  \Gin@nat@height\Gin@ury bp%
  \advance\Gin@nat@height-\Gin@lly bp%
```

```
\Gin@nat@width\Gin@urx bp%
\advance\Gin@nat@width-\Gin@llx bp%
\Gin@req@sizes
\expandafter\ifx\csname Ginclude@#1\endcsname\relax
  \Gin@drafttrue
  \expandafter\ifx\csname Gread@#1\endcsname\relax
    \@latex@error{Can not include graphics of type: #1}\@ehc
    \global\expandafter\let\csname Gread@#1\endcsname\@empty
  \fi
\fi
\leavevmode
```

Here the tagging begins. We want to catch also the draft box, and for luatex tagging must be started before the `\setbox`.

```
\tag_socket_use:ne {graphic/init} {tagging-setup={mode=\l_tag_graphic_mode_tl}}
\tag_socket_use:nn {graphic/begin} {\Gin@base\Gin@ext}
```

We store also the draft box in a box and do not output it directly so that we can calculate its BBox too.

```
\ifGin@draft
   \setbox\z@
    \hb@xt@\Gin@req@width{%
      \vrule\hss
      \vbox to \Gin@req@height{%
         \hrule \@width \Gin@req@width
         \vss
         \edef\@tempa{#3}%
         \rlap{ \ttfamily\expandafter\strip@prefix\meaning\@tempa}%
         \vss
         \hrule}%
      \hss\vrule}%
\else
  \@addtofilelist{#3}%
  \ProvidesFile{#3}[Graphic~file~(type~#1)]%
  \setbox\z@\hbox{\csname Ginclude@#1\endcsname{#3}}%
  \dp\z@\z@
  \ht\z@\Gin@req@height
  \wd\z@\Gin@req@width
\fi
```

This ends the tagging.

```
\tag_socket_use:nnn{graphic/end}
  { \Gin@tag@bbox@attribute }
  { \box\z@ }
 }
}
```

### 7.5.3 Calculating the BBox

This is the large code part.

__tag_graphic_get_trim:

Graphics can be trimmed with the trim and the viewport key. If the graphic is not clipped the values must be taken into account when rotating. If viewport is used we have to calculate the trim.

```
\cs_new_protected:Npn \__tag_graphic_get_trim:
 {
   \legacy_if:nTF {Gin@clip}
```

Setting to 0 is not strictly needed but looks cleaner.

```
    {
      \fp_zero:N\l__tag_graphic_trim_lx_fp
      \fp_zero:N\l__tag_graphic_trim_ly_fp
      \fp_zero:N\l__tag_graphic_trim_ux_fp
      \fp_zero:N\l__tag_graphic_trim_uy_fp
    }
    {
      \fp_set:Nn \l__tag_graphic_trim_lx_fp {\l__tag_graphic_scale_fp*\Gin@vllx}
      \fp_set:Nn \l__tag_graphic_trim_ly_fp {\l__tag_graphic_scale_fp*\Gin@vlly}
      \fp_set:Nn \l__tag_graphic_trim_ux_fp {\l__tag_graphic_scale_fp*\Gin@vurx}
      \fp_set:Nn \l__tag_graphic_trim_uy_fp {\l__tag_graphic_scale_fp*\Gin@vury}
      \cs_if_exist:NT \Gin@ollx
        {
         \fp_set:Nn \l__tag_graphic_trim_ux_fp {\l__tag_graphic_scale_fp* (\Gin@ourx-(\Gin@urx)) }
         \fp_set:Nn \l__tag_graphic_trim_uy_fp {\l__tag_graphic_scale_fp* (\Gin@oury-(\Gin@ury)) }
        }
    }
 }
```

(*End of definition for* `\__tag_graphic_get_trim:`.)

__tag_graphic_get_scale:

```
\cs_new_protected:Npn \__tag_graphic_get_scale:
 {
   \fp_set:Nn \l__tag_graphic_scale_fp
     {
       \str_if_eq:eeTF {\Gin@scalex} { ! }
        { \Gin@scaley }
        { \Gin@scalex }
     }
 }
```

(*End of definition for* `\__tag_graphic_get_scale:`.)

__tag_graphic_applyangle:nnnn

This takes the current BBox and rotates it according to the use angle. This is the most laborious code, as we have to take also the trim values into account. We have to compare the values after the rotation to find the right corners for the BBox. Not sure, if this is the most effective code, the l3draw package has similar code to calculate a rotation, this can perhaps be reused ...

```
\cs_new_protected:Npn \__tag_graphic_applyangle:nnnn #1#2#3#4 %lx,ly,ux,uy
 {
   \bool_lazy_and:nnT
     {\cs_if_exist_p:N \Grot@angle }
     {! \int_compare_p:nNn { \Grot@angle }={0}}
```

```
    {
      \fp_set:Nn \l__tag_graphic_sin_fp  { sind(\Grot@angle) }
      \fp_set:Nn \l__tag_graphic_cos_fp  { cosd(\Grot@angle) }
      \fp_set:Nn \l__tag_graphic_lx_fp {#1}
      \fp_set:Nn \l__tag_graphic_ly_fp {#2}
      \fp_set:Nn \l__tag_graphic_ux_fp {#3}
      \fp_set:Nn \l__tag_graphic_uy_fp {#4}
```

get the x coordinates (cos,-sin)

```
      \fp_set:Nn\l__tag_graphic_lxly_fp
       {
         -\l__tag_graphic_trim_lx_fp *  \l__tag_graphic_cos_fp
         +\l__tag_graphic_trim_ly_fp *  \l__tag_graphic_sin_fp
       }
      \fp_set:Nn\l__tag_graphic_lxuy_fp
       {
         (-\l__tag_graphic_trim_lx_fp) * \l__tag_graphic_cos_fp
         +
          (\l__tag_graphic_uy_fp-\l__tag_graphic_ly_fp-\l__tag_graphic_trim_ly_fp)
           * (-\l__tag_graphic_sin_fp)
       }
      \fp_set:Nn\l__tag_graphic_uxly_fp
       {
         (\l__tag_graphic_ux_fp-\l__tag_graphic_lx_fp-\l__tag_graphic_trim_lx_fp)
          * \l__tag_graphic_cos_fp
         +
         (\l__tag_graphic_trim_ly_fp) * (\l__tag_graphic_sin_fp)
       }
      \fp_set:Nn\l__tag_graphic_uxuy_fp
       {
         (\l__tag_graphic_ux_fp-\l__tag_graphic_lx_fp-\l__tag_graphic_trim_lx_fp)
           *  \l__tag_graphic_cos_fp
        +
         (\l__tag_graphic_uy_fp-\l__tag_graphic_ly_fp-\l__tag_graphic_trim_ly_fp)
           * (-\l__tag_graphic_sin_fp)
       }
      \tl_gset:Ne\g__tag_graphic_lx_tl
        {
         \fp_eval:n
           {
             min
              (
                \l__tag_graphic_lxly_fp,
                \l__tag_graphic_lxuy_fp,
                \l__tag_graphic_uxly_fp,
                \l__tag_graphic_uxuy_fp,
              )
              +\l__tag_graphic_lx_fp
              +\l__tag_graphic_trim_lx_fp
           }
        }
      \tl_gset:Ne\g__tag_graphic_ux_tl
        {
          \fp_eval:n
```

```
380             {
381               max
382                (
383                  \l__tag_graphic_lxly_fp,
384                  \l__tag_graphic_lxuy_fp,
385                  \l__tag_graphic_uxly_fp,
386                  \l__tag_graphic_uxuy_fp
387                )
388                +\l__tag_graphic_lx_fp
389                +\l__tag_graphic_trim_lx_fp
390             }
391          }
```

get the y coordinates (sin,cos)

```
392        \fp_set:Nn\l__tag_graphic_lxly_fp
393         {
394            -\l__tag_graphic_trim_lx_fp *  \l__tag_graphic_sin_fp
395            -\l__tag_graphic_trim_ly_fp *  \l__tag_graphic_cos_fp
396         }
397        \fp_set:Nn\l__tag_graphic_lxuy_fp
398         {
399           - \l__tag_graphic_trim_lx_fp *  \l__tag_graphic_sin_fp
400           +
401           (\l__tag_graphic_uy_fp-\l__tag_graphic_ly_fp-\l__tag_graphic_trim_ly_fp)
402             * \l__tag_graphic_cos_fp
403         }
404        \fp_set:Nn\l__tag_graphic_uxly_fp
405         {
406           (\l__tag_graphic_ux_fp-\l__tag_graphic_lx_fp-\l__tag_graphic_trim_lx_fp)
407             * \l__tag_graphic_sin_fp
408           - \l__tag_graphic_trim_ly_fp * \l__tag_graphic_cos_fp
409         }
410        \fp_set:Nn\l__tag_graphic_uxuy_fp
411         {
412           (\l__tag_graphic_ux_fp-\l__tag_graphic_lx_fp-\l__tag_graphic_trim_lx_fp)
413             * \l__tag_graphic_sin_fp
414          +
415           (\l__tag_graphic_uy_fp-\l__tag_graphic_ly_fp-\l__tag_graphic_trim_ly_fp)
416             * \l__tag_graphic_cos_fp
417         }
418        \tl_gset:Ne\g__tag_graphic_ly_tl
419          {
420           \fp_eval:n
421             {
422               min
423                (
424                  \l__tag_graphic_lxly_fp,
425                  \l__tag_graphic_lxuy_fp,
426                  \l__tag_graphic_uxly_fp,
427                  \l__tag_graphic_uxuy_fp
428                )
429               + \l__tag_graphic_ly_fp + \l__tag_graphic_trim_ly_fp
430             }
431          }
```

```
      \tl_gset:Ne\g__tag_graphic_uy_tl
        {
          \fp_eval:n
            {
              max
               (
                 \l__tag_graphic_lxly_fp,
                 \l__tag_graphic_lxuy_fp,
                 \l__tag_graphic_uxly_fp,
                 \l__tag_graphic_uxuy_fp,
               )
              + \l__tag_graphic_ly_fp + \l__tag_graphic_trim_ly_fp
            }
        }
    }
}
\cs_generate_variant:Nn\__tag_graphic_applyangle:nnnn {VVVV}
```

(*End of definition for* `\__tag_graphic_applyangle:nnnn`.)

`\__tag_graphic_applycorr:NNNN` This command is used to add at the end the correction values. Quite dump ...

```
\cs_new_protected:Npn \__tag_graphic_applycorr:NNNN #1 #2 #3 #4
{
 \bool_if:NT\l__tag_graphic_bboxcorr_bool
   {
    \tl_set:Ne #1
      {
        \fp_eval:n
         {
          #1
          +
          \dim_to_decimal_in_bp:n {\seq_item:Nn \l__tag_graphic_bboxcorr_seq {1} }
         }
      }
    \tl_set:Ne #2
      {
        \fp_eval:n
         {
          #2
          +
          \dim_to_decimal_in_bp:n {\seq_item:Nn \l__tag_graphic_bboxcorr_seq {2} }
         }
      }
    \tl_set:Ne #3
      {
        \fp_eval:n
         {
          #3
          +
          \dim_to_decimal_in_bp:n {\seq_item:Nn \l__tag_graphic_bboxcorr_seq {3} }
         }
      }
    \tl_set:Ne #4
      {
        \fp_eval:n
```

```
          {
           #4
           +
           \dim_to_decimal_in_bp:n {\seq_item:Nn \l__tag_graphic_bboxcorr_seq {4} }
          }
       }
    }
}
```

(*End of definition for* `\__tag_graphic_applycorr:NNNN`.)

`\Gin@tag@bbox@attribute` This is the main command to calculate and set the Bbox attribute of the `\includegraphics` command. It also sets the reference point on the page.

```
\cs_new_protected:Npn \Gin@tag@bbox@attribute
  {
    \__tag_graphic_get_scale:
    \__tag_graphic_get_trim:
    \int_gincr:N\g__tag_graphic_int
    \tl_set:Ne\l__tag_graphic_currentlabel_tl {__tag_graphic.\int_use:N \g__tag_graphic_int}
    \__tag_graphic_savepos:e { \l__tag_graphic_currentlabel_tl }
    \tl_gset:Ne\g__tag_graphic_lx_tl
      {
        \dim_to_decimal_in_bp:n
          { \property_ref:een {\l__tag_graphic_currentlabel_tl}{xpos}{0}sp }
      }
    \tl_gset:Ne\g__tag_graphic_ly_tl
      {
        \dim_to_decimal_in_bp:n
          { \property_ref:een {\l__tag_graphic_currentlabel_tl}{ypos}{0}sp }
      }
    \tl_gset:Ne\g__tag_graphic_ux_tl
        {
          \fp_eval:n
           {
             \g__tag_graphic_lx_tl
             +
             \dim_to_decimal_in_bp:n { \Gin@req@width }
           }
        }
     \tl_gset:Ne\g__tag_graphic_uy_tl
        {
          \fp_eval:n
           {
             \g__tag_graphic_ly_tl
             +
              \dim_to_decimal_in_bp:n  { \Gin@req@height }
           }
        }
```

If the graphics is not clipped we must add the trim values.

```
    \legacy_if:nF {Gin@clip}
      {
        \tl_gset:Ne\g__tag_graphic_ux_tl
          {
```

```
            \fp_eval:n
              {
                \g__tag_graphic_ux_tl
                +
                \l__tag_graphic_trim_ux_fp
              }
          }
        \tl_gset:Ne\g__tag_graphic_lx_tl
          {
            \fp_eval:n
              {
                \g__tag_graphic_lx_tl
                 -
                \l__tag_graphic_trim_lx_fp
              }
          }
        \tl_gset:Ne\g__tag_graphic_uy_tl
          {
            \fp_eval:n
              {
                \g__tag_graphic_uy_tl
                +
                \l__tag_graphic_trim_uy_fp
              }
          }
        \tl_gset:Ne\g__tag_graphic_ly_tl
          {
            \fp_eval:n
              {
                \g__tag_graphic_ly_tl
                -
                \l__tag_graphic_trim_ly_fp
              }
          }
      }
```

If there is an angle we now rotate the values.

```
    \__tag_graphic_applyangle:VVVV
      \g__tag_graphic_lx_tl
      \g__tag_graphic_ly_tl
      \g__tag_graphic_ux_tl
      \g__tag_graphic_uy_tl
```

At last we have to add the correction values

```
    \__tag_graphic_applycorr:NNNN
      \g__tag_graphic_lx_tl
      \g__tag_graphic_ly_tl
      \g__tag_graphic_ux_tl
      \g__tag_graphic_uy_tl

  \bool_if:NT\l__tag_graphic_debug_bool
   {
     \__tag_graphic_show_bbox:VVVVne
      \g__tag_graphic_lx_tl
      \g__tag_graphic_ly_tl
```

```
      \g__tag_graphic_ux_tl
      \g__tag_graphic_uy_tl
      {red}
      {\l__tag_graphic_currentlabel_tl}
  }
```

Now we add the attribute. We do it manually as it had to be delayed until now. The structure and the mc must be open earlier, before the `\setbox` (at least for luatex it has to).

```
   \tag_struct_gput:ene{\tag_get:n{struct_num}}{attribute}
      {
        /O /Layout /BBox~
          [
           \g__tag_graphic_lx_tl\c_space_tl
           \g__tag_graphic_ly_tl\c_space_tl
           \g__tag_graphic_ux_tl\c_space_tl
           \g__tag_graphic_uy_tl
          ]
      }
}
```

(*End of definition for* `\Gin@tag@bbox@attribute`.)

## 7.6 Support for the picture environment

### 7.6.1 User interface

The original picture has no key-val argument yet. In the new optional argument uses the generic tag/graphic keys. We could perhaps use the `Gin` keys instead, but there could be side-effects if some uses other `Gin` keys like `angle`, so better stay on the safe side.

### 7.6.2 Calculation of the BBox

`\picture@tag@bbox@attribute`

Picture needs a similar command to calculate the BBox. But here we stay simple and use simply the size of the picbox.

```
\newcommand\picture@tag@bbox@attribute
 {
   \int_gincr:N\g__tag_graphic_int
   \tl_set:Ne\l__tag_graphic_currentlabel_tl {__tag_graphic.\int_use:N \g__tag_graphic_int}
   \__tag_graphic_savepos:e { \l__tag_graphic_currentlabel_tl }
   \tl_gset:Ne \g__tag_graphic_lx_tl
     {
       \dim_to_decimal_in_bp:n
       { \property_ref:een {\l__tag_graphic_currentlabel_tl}{xpos}{0}sp  }
     }
   \tl_gset:Ne \g__tag_graphic_ly_tl
     {
       \dim_to_decimal_in_bp:n
       { \property_ref:een {\l__tag_graphic_currentlabel_tl}{ypos}{0}sp - \dp\@picbox }
     }
   \tl_gset:Ne \g__tag_graphic_ux_tl
     {
       \dim_to_decimal_in_bp:n
        {
```

```
          \g__tag_graphic_lx_tl bp + \wd\@picbox
        }
    }
  \tl_gset:Ne \g__tag_graphic_uy_tl
    {
      \dim_to_decimal_in_bp:n
       {
         \g__tag_graphic_ly_tl bp + \ht\@picbox + \dp\@picbox
       }
    }
  \__tag_graphic_applycorr:NNNN
       \g__tag_graphic_lx_tl
       \g__tag_graphic_ly_tl
       \g__tag_graphic_ux_tl
       \g__tag_graphic_uy_tl
  \bool_if:NT\l__tag_graphic_debug_bool
    {
      \__tag_graphic_show_bbox:VVVVne
       \g__tag_graphic_lx_tl
       \g__tag_graphic_ly_tl
       \g__tag_graphic_ux_tl
       \g__tag_graphic_uy_tl
       {red}
       {\l__tag_graphic_currentlabel_tl}
    }
```

this stores the attribute in the structure.

```
  \tag_struct_gput:ene{\tag_get:n{struct_num}}{attribute}
        {
            /O /Layout /BBox~
            [
             \g__tag_graphic_lx_tl\c_space_tl
             \g__tag_graphic_ly_tl\c_space_tl
             \g__tag_graphic_ux_tl\c_space_tl
             \g__tag_graphic_uy_tl
            ]
        }
}
```

(*End of definition for* `\picture@tag@bbox@attribute`.)

### 7.6.3 Patching the commands

We redefine `\picture` to accept an optional argument. We also ensure that we are in hmode, so that stopping tagging doesn't confuse the paratags.

```
\RenewDocumentCommand\picture{O{}m}
 {
   \leavevmode
   \tag_socket_use:nn{graphic/init}{#1}
   \pictur@#2
 }
```

inside the picture box we stop tagging.

```
\def\@picture(#1,#2)(#3,#4){%
```

```
  \@defaultunitsset\@picht{#2}\unitlength
  \@defaultunitsset\@tempdimc{#1}\unitlength
  \tag_socket_use:nn{graphic/begin}{picture~environment}
  \tag_suspend:n{\@picture} %do not tag inside the picture box
  \setbox\@picbox\hb@xt@\@tempdimc\bgroup
    \@defaultunitsset\@tempdimc{#3}\unitlength
    \hskip -\@tempdimc
    \@defaultunitsset\@tempdimc{#4}\unitlength
    \lower\@tempdimc\hbox\bgroup
      \ignorespaces}

\def\endpicture{%
  \egroup\hss\egroup
    \ht\@picbox\@picht\dp\@picbox\z@
    \tag_resume:n{\@picture}
    \tag_socket_use:nnn{graphic/end}
      {\picture@tag@bbox@attribute}
      {\mbox{\box\@picbox}}}
```

## 7.7 Debugging code

This command puts a transparent layer in the size of the BBox over an graphic.

__tag_graphic_show_bbox:nnnnnn

```
\cs_new_protected:Npn \__tag_graphic_show_bbox:nnnnnn #1#2#3#4#5#6%#5 color, #6 label name
 {
    \iow_log:n {tag/graphic~debug:~BBox~of~graphics~#6~is~#1~#2~#3~#4}
    \hook_gput_code:nnn
     {shipout/foreground}
     {tag/graphic}
     {
      \int_compare:nNnT
       {\g_shipout_readonly_int}
       =
       {\property_ref:een{#6}{abspage}{0}}
       {
        \put
         (#1 bp,\dim_eval:n{-\paperheight + \dim_eval:n{#2 bp}})
         {
           \opacity_select:n{0.5}\color_select:n{#5}
           \SuspendTagging{BBox}   % as long as the debug code this isn't tagged
           \rule
               {\dim_eval:n {#3 bp-\dim_eval:n{#1 bp}}}
               {\dim_eval:n {#4 bp-\dim_eval:n{#2 bp}}}
         }
       }
     }
 }
 \cs_generate_variant:Nn  \__tag_graphic_show_bbox:nnnnnn {VVVVne,nnnnne}
```

(*End of definition for* `\__tag_graphic_show_bbox:nnnnnn`.)

```
⟨/package⟩
```