The **tikzmark** package

Andrew Stacey
`loopspace@mathforge.org`

v1.6 from 2018/10/18

1 Introduction

The `\tikzmark` macro burst onto the scene in a blaze of glory on TeX-SX. Since then, it has proved embarrassingly (to its original author) popular. The idea behind it is extremely simple: that the machinery underneath TikZ provides a way to "mark" a point on a page for further use. This functionality is already provided by several other packages. The point of this one is that as TikZ can provide this feature, if already loading TikZ then it makes sense to use the TikZ version than another version. Moreover, if the goal is to use these marks with some TikZ code then this version is already set up for that purpose (not that it would be exactly difficult to add this to any of the other implementations).

2 Use

Using the `\tikzmark` is extremely simple. You need to load the `tikz` package and then load `tikzmark` as a `tikzlibrary`. Thus in your preamble you should have something like:

```
\usepackage{tikz}
\usetikzlibrary{tikzmark}
```

In your document, you can now type `\tikzmark{<name>}` at a point that you want to remember. This will save a mark with name `<name>` for use later (or earlier). To use it in a `\tikz` or `tikzpicture`, simply use the `pic` coordinate system:

```
\tikz[remember picture] \draw[overlay] (0,0) -- (pic
    cs:<name>);
```

There are two important points to note:

1. The enveloping `\tikz` or `tikzpicture` must have the key `remember picture` set.

   This is because of how TikZ coordinates work. The coordinates inside a TikZ picture are relative to its origin, so that origin can move around on

the page and not affect the internals of the picture. To use a point outside the picture, therefore, the current picture not only has to know where that point is on the page it also has to know where it itself is on the page. Hence the `remember picture` key must be set.

2. The drawing command must have the `overlay` key set (or be in a scope or picture where it is set).

   This is to keep the bounding box of the current picture under control. Otherwise, it would grow to encompass the remembered point as well as the current picture. (This isn't necessary if the remembered point is inside the current picture.)

3   History

I wrote the original `\tikzmark` macro in 2009 for use in lecture slides prepared with the `beamer` package. Its original definition was:

```
\newcommand{\tikzmark}[1]{\tikz[overlay,remember picture]
    \node (#1) {};}
```

Its first use was in the (inelegant) code:

```
\begin{frame}
\frametitle{Structure of Continuous Functions}

\begin{tikzpicture}[overlay, remember picture]
\useasboundingbox (0,0);
\draw<2-|trans: 0|handout: 0>[red,->] (bsp) .. controls
    +(-1,-1) and ($(cnvs.north)+(1,1)$) ..
    ($(cnvs.north)+(0,1)$) .. controls
    ($(cnvs.north)+(-1,1)$) and +(-1,0) .. (cnvs.north);
\draw<3-|trans: 0|handout: 0>[green!50!black,->] (cplt) ..
    controls +(-1,-1) and +(-1,0) .. (mcplt.north);
\draw<4-|trans: 0|handout: 0>[blue,->] (norm) .. controls
    +(-1,-.5) and ($(nvs.north)+(0,1.5)$) ..
    ($(nvs.north)+(0,1.5)$) .. controls
    ($(nvs.north)+(-1.5,1.5)$) and +(-1.5,0) ..
    (nvs.north);
\draw<5-|trans: 0|handout: 0>[purple,->] (vector) ..
    controls +(-1,-1) and ($(vsp.north)+(2,2)$) ..
    ($(vsp.north)+(0,2)$) .. controls
    ($(vsp.north)+(-2,2)$) and +(-2,0) .. (vsp.north);
\end{tikzpicture}

\begin{theorem}
\centering
\(\big(C([0,1],\R),d_\infty\big)\) \\
```

```
is a \\
\alert{Banach\tikzmark{bsp} space}
\end{theorem}

\pause
\bigskip

\begin{itemize}
\item[\tikzmark{cnvs}]
    {\color<.(2)->{green!50!black}Comp\tikzmark{cplt}lete}
    {\color<.(3)->{blue}nor\tikzmark{norm}med}
    {\color<.(4)->{purple}vector\tikzmark{vector} space}.

\bigskip
\bigskip
\pause

\begin{itemize}[<+->]
\item[\tikzmark{mcplt}] {\color{green!50!black}Cauchy
    sequences converge.}
\medskip
\item[\tikzmark{nvs}] {\color{blue}Metric from a norm.}
\medskip
\item[\tikzmark{vsp}] {\color{purple}Functions behave like
    vectors.}
\end{itemize}
\end{itemize}

\end{frame}
```

This produced, on the final slide, Figure 1.

Its first appearance on TeX-SX was in an answer to a question about how to put overlapping braces on a mathematical text. This was in July 2010. The opening statement of the answer was not overly encouraging: "This may not be the best solution. . .". And for a macro that would go on to become quite ubiquitous, its initial appearance only garnered it 2 votes.

However, it started out in life as a useful macro for me and as such I found more uses for it in my own code and thus more opportunity for using it to answer questions on TeX-SX. The one that seems to have been where it got noticed came in August 2010, again about putting braces in text but in a more complicated fashion. From this answer, it got picked up, picked over, and picked apart. A common use was in highlighting or adding marks to text.

Gradually, as it got used, it developed. A major revision dates from an answer given in March 2012 where the question was actually about \tikzmark. This version added two important features: a TikZ coordinate system for referencing saved marks directly and the ability to refer to marks earlier in the document

Structure of Continuous Functions

Theorem

$$\Big(C([0,1],\mathbb{R}),d_\infty\Big)$$
is a
Banach space

▸ Complete normed vector space.

▸ Cauchy sequences converge.

▸ Metric from a norm.

▸ Functions behave like vectors.

Figure 1: First use of tikzmark

than they are defined (the mechanism for remembering points uses the `aux` file anyway so this was more about exposing the information earlier than anything complicated). Then in October 2012 there was a question where it would have been useful to remember which page the mark was on and a question where for some reason using the `\tikz` macro didn't work so the `\pgfmark` macro was introduced.

By this point, the `\tikzmark` command had morphed considerably from its original definition. Experience has shown that on the TeX-SX site it has continued to be used in its original form as well as its current form. I've therefore taken the decision to reintroduce a form of the original command, now called `\tikzmarknode`. It goes beyond the original version in that it uses some `\mathchoice` trickery (inspired by this answer from Heiko Oberdiek) to hopefully correctly choose the correct math style.

The original reason for not using nodes inside `\tikzmark` was to be able to use the information from a `\tikzmark` before the point where it was defined (via information saved into the `aux` file). Thanks to a question on TeX-SX about saving node information, I've developed code that solves that issue with nodes. As it fits in the general concept of this package, I've added that code to the `\tikzmark` package.

4   Usage

This package defines the following commands and usable stuff.

1. `\tikzmark[`⟨*drawing command*⟩`]{`⟨*name*⟩`}`

   The mandatory argument is the name of the mark to be used to refer back to this point later.

   The `\tikzmark` command can take an optional parameter which is some drawing command that can be put in a `\tikz ... ;` command. This drawing command can be used to place a node or something similar at the marked point, or to set some `\tikzset` keys. Sometimes this can be useful. Note, though, that if this is used to define an offset coordinate then this will only be available in the document *after* the `\tikzmark` command, even on later runs.

   If the `beamer` class is loaded then this command is made overlay-aware.

2. `\tikzmark{`⟨*name*⟩`}{`⟨*coordinate*⟩`}`

   v1.2 of the `tikzmark` package introduced a new variant of `\tikzmark` which works inside a `tikzpicture`. One feature of `\tikzmark` which isn't part of TikZ's normal coordinate remembering system is the ability to use a `\tikzmark` coordinate before it is defined (due to the use of the `aux` file). This is potentially useful to have inside a `tikzpicture` and so it is now possible to use `\tikzmark` inside a `tikzpicture`. The syntax is slightly different as we need to specify the coordinates of a point to remember.

   This was inspired by the question Refer to a node in tikz that will be defined "in the future" (two passes)? on TeX-SX.

3. `\pgfmark{⟨name⟩}`

This is a more basic form of the `\tikzmark` which doesn't use any of the `\tikz` overhead. One advantage of this command is that it doesn't create an `hbox`. It does, however, insert a `whatsit` into the stream so it will, for example, stop two vertical spaces either side of it being merged. This can't be avoided.

If the `beamer` class is loaded then this command is made overlay-aware.

4. `\iftikzmark{⟨name⟩}{⟨true code⟩}{⟨false code⟩}`

This is a simple conditional to test if a particular mark is available. It executes `true code` if it is and `false code` if not.

5. `\tikzmarknode[⟨options⟩]{⟨name⟩}{⟨contents⟩}`

This is a reincarnation of the original `\tikzmark` command which places its contents inside a `\tikz` node. It also defines a `tikzmark` with the same name. Using a sneaky trick with `\mathchoice`, it works inside a math environment. The spacing either side might not be quite right as although it detects the math style it doesn't got beyond that. The `options` are passed to the node.

Two styles are attempted, one on the surrounding picture and one on the node, which are:

   • `every tikzmarknode picture`
   • `every tikzmarknode`

To refer to the *node*, use usual TikZ coordinates. To refer to the underlying *tikzmark*, use the special tikzmark coordinates (see below).

6. `(pic cs:<name>)` or `(pic cs:<name>,<coordinate>)`

This is the method for referring to a position remembered by `\tikzmark` (or `\pgfmark`) as a coordinate in a `tikzpicture` environment (or `\tikz` command). If the extra `coordinate` is specified then this is used in case the mark `name` has not yet been defined (this can be useful for defining code that does something sensible on the first run).

7. `/tikz/save picture id=<name>`

This is the TikZ key that is used by `\tikzmark` to actually save the connection between the name and the picture coordinate. It can be used on an arbitrary picture to save its origin as a tikzmark.

8. `/tikz/if picture id=#1#2#3`

This is a key equivalent of the `\iftikzmark` command.

9. `/tikz/next page`, `/tikz/next page vector`

It is possible to refer to a mark on a different page to the current page. When this is done, the mark is offset by a vector stored in the key

`/tikz/next page vector`. The key `/tikz/next page` can be used to set this to certain standard vectors by specifying where the "next page" is considered as lying corresponding to the current page. Possible values are (by default) `above`, `below`, `left`, `right`, and `ignore`. (The last one sets the vector to the zero vector.)

10. `/tikz/tikzmark prefix=<prefix>` and `/tikz/tikzmark suffix=<suffix>`

   These keys allow for the automatic addition of a prefix and/or suffix to each `\tikzmark` name. The prefix and suffix are added both at time of definition and of use, so providing one is in the same scope there is no difference in at the user level when using prefixes and suffixes. What it can be useful for is to make the `\tikzmark` names unique. In particular, if the `beamer` class is loaded then an automatic suffix is added corresponding to the overlay. This means that if a slide consists of several overlays with `\tikzmark`s on them, and the positions of the `\tikzmark`s move then the resulting pictures should look right. Without the automatic suffix, only the final positions of the marks would be used throughout.

   This was inspired by the question using tikzmark subnode with overlays beamer on TeX-SX.

11. `\subnode[options]{name}{content}`

   This produces a pseudo-node named `name` around the `content`. The design purpose of this is to create a "subnode" inside a TikZ node. As far as TikZ is concerned, the contents of a node is just a box. It therefore does not know anything about it beyond its external size and so cannot easily determine the coordinates of pieces inside. The `\subnode` command boxes its contents and saves the position of that box and its dimensions. This information is stored in the same way that PGF stores the necessary information about a node. It is therefore possible to use ordinary node syntax (within a `tikzpicture`) to access this information. Thus after `\node {a \subnode{a}{sub} node};` it is possible to use `a` as a node. The `options` are passed to the node construction mechanism, but note that the only sensible options are those that affect the size and shape of the node: drawing options are ignored (except in so far as they affect the size – as an example, `line width` affects the node size).

   There are two important points to make about this. The first is that, as with all the `tikzmark` macros, the information is always one compilation old. The second is that the pseudo-node is purely about coordinates: the path information is not used and the contents are not moved. This is partly for reasons of implementation: the pseudo-node is constructed when TikZ is not in "picture mode". But also interleaving the background path of the pseudo-node and any containing node would be problematic and so is best left to the user.

   The simplest way to turn a pseudo-node into a more normal node is to use the `fit` library. Using the above example, `\node[fit=(a),draw,inner sep=0pt] {};`

7

would draw a rectangle around the word `sub` of exactly the same size as would appear had a normal node been created.

12. *Node saving*

The node saving system takes the information stored about a node and saves it for later use. That later use can be in the same document, in which case it should be saved just to the memory of the current TeX process, or it can be used earlier in the same document or another document altogether (in particular, if the nodes are defined in a `tikzpicture` that has been externalised, this can be used to import the node information into the main file) in which cases the node data is saved to a file.

When working with files, nodes are saved and restored in bulk. When working in memory, nodes are saved and restored in named lists. Nodes are not actually saved until the end of the tikzpicture in which they are defined, meaning that if saving to memory then all the nodes in a tikzpicture will belong to the same list.

The keys for working with saving and restoring nodes are as follows.

- `save node`

  This is the key to put on a node that is to be saved.

- `set node group=<name>`

  Nodes are grouped together into a list that can be saved either to a file or for use later on in the document. This sets the name for the current group.

- `restore nodes from list=<name>`

  This restores the node information from the named list to the current `tikzpicture`. This is required both for when the node information comes from a file or from earlier in the same document.

- `save nodes to file`

  This is a `true/false` key which determines whether to save the node information to a file.

- `set saved nodes file name=<name>`

  This sets the file name for the saved nodes (the extension will be `.nodes`. The default is to use the current TeX filename. This is set globally, and once the file is opened then changing the name will have no effect. (The file is not opened until it is actually needed to avoid creating empty files unnecessarily.)

- `restore nodes from file=<name>`

  This loads the node information from the file into the current document. The `<name>` can have the syntax `[options]{name}`, where `options` can be used to influence how the nodes are restored. The key `transform saved nodes` (see below) can be given here. Another useful key is the `name prefix` key which is applied to all restored nodes.

- `transform saved nodes`

  A particular use-case for restoring saved nodes is to safely include one `tikzpicture` inside another by creating an image out of the inner picture and including it back in as a picture inside a node. In that situation, restoring the nodes from the inner picture can make it possible to refer to coordinates from the inner picture to the outer one. If there is a transformation in place on the containing node, this key applies that transformation to all the nodes in the inner picture.
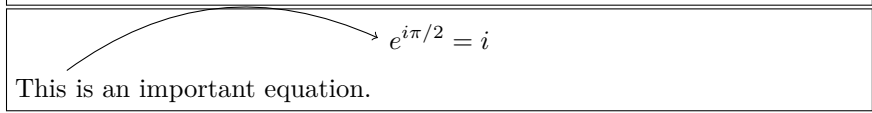
5 Examples and Extras

The `\tikzmark` command has been used in numerous answers on TeX-SX. The plan is to gather some of these into extra libraries which can be loaded via `\usetikzmarklibrary`.

At present, this is the code listings library (which works with the `listings` package). One that is in development (as it has featured much on the TeX-SX website) is highlighting, however this is not so straightforward to implement so is still under development.

5.1 Basic Examples

A simple example of the `\tikzmark` macro is the following.

```
\[
 \tikzmark{a} e^{i \pi/2} = i
\]

This\tikz[remember picture,overlay,baseline=0pt]
    \draw[->] (0,1em) to[bend left]
    ([shift={(-1ex,1ex)}]pic cs:a); is an important
    equation.
```

$e^{i\pi/2} = i$

This is an important equation.

```
\begin{itemize}
\item A first item,\tikzmark{b}
\item A second item,\tikzmark{c}
\item A third item.\tikzmark{d}
\end{itemize}
\begin{tikzpicture}[remember picture,overlay]
\draw[decorate,decoration={brace}] ({pic cs:c} |- {pic
    cs:b}) +(0,1em) -- node[right,inner sep=1em] {some
    items} ({pic cs:c} |- {pic cs:d});
\end{tikzpicture}
```

- A first item,
- A second item, some items
- A third item.

```
\begin{tikzpicture}[remember picture]
\node (a) at (0,0) {This has a \subnode{sub}{subnode} in
    it};
\draw[->] (0,-1) to[bend right] (sub);
\end{tikzpicture}
```

This has a subnode in it

An example using \tikzmark inside a tikzpicture

```
\tikzset{tikzmark prefix=ex3-}
\begin{tikzpicture}[remember picture,overlay]
\draw[->,line width=1mm,cyan] (pic cs:a) to[bend left]
    (pic cs:b);
\end{tikzpicture}

By placing the \tikzmark{a}code before the marks, the
    arrow goes under the subsequent text and picture.

\begin{tikzpicture}
\filldraw[fill=gray] (0,0) circle[radius=1cm];
\tikzmark{b}{(-1,-1)}
\end{tikzpicture}
```
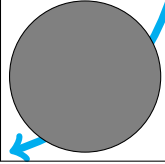
By placing the code before the marks, the arrow goes under the subsequent text and picture.

The \tikmarknode puts a node around some text, which can be referred to later, and adds a \tikzmark at its origin.

```
Putting a node around \tikzmarknode{txt}{some text}
    means we can connect text together, including in
    maths:
\[
\tikzmarknode{a}{\sum_{k=1}^n} k^{\tikzmarknode{b}{2}}
\]

\begin{tikzpicture}[remember picture,overlay]
\draw[->] (txt) -- (a);
\draw[->] (a.south) to[out=-90,in=-45] (b.south east);
\end{tikzpicture}
```

Putting a node around some text means we can connect text together, including in maths:
$$\sum_{k=1}^{n} k^2$$

The syntax for saving node data is illustrated by the following example. File `firstpicture.tex`:

```
\documentclass[tikz,border=10pt]{standalone}
\usetikzlibrary{tikzmark,shapes.geometric}
\begin{document}
\begin{tikzpicture}[save nodes to file]
\node[draw,rotate=-30,save node](1) at (-2,0) {1};
\draw[->] (0,0) -- (1);
\node[draw,ellipse,save node] (c) at (current bounding
    box.center) {};
\end{tikzpicture}
\end{document}
```

File `secondpicture.tex`:

```
\documentclass[tikz,border=10pt]{standalone}
\usetikzlibrary{tikzmark,shapes.geometric}
\begin{document}
\begin{tikzpicture}[save nodes to file]
\node[draw,rotate=-70,save node] (2) at (2,0) {2};
\draw[->] (0,0) -- (2);
\node[draw,ellipse,save node] (c) at (current bounding
    box.center) {};
\end{tikzpicture}
\end{document}
```

Main file:

```
\documentclass{article}
\usepackage{tikz}
\usetikzlibrary{tikzmark}

\begin{document}
\begin{tikzpicture}

\node[draw,
  rotate=30,
  restore nodes from file={[transform saved nodes,name
      prefix=pic-1-]{firstpicture}}
] (a-1) at (-2,-3) {\includegraphics{firstpicture.pdf}};

\node[draw,
  rotate=70,
  restore nodes from file={[transform saved nodes,name
      prefix=pic-2-]{secondpicture}}
] (a-2) at (+2,+2) {\includegraphics{secondpicture.pdf}};

\draw[red] (pic-1-1.north west) -- (pic-1-1.north east)
    -- (pic-1-1.south east) -- (pic-1-1.south west) --
    cycle;
\draw[red] (pic-2-2.north west) -- (pic-2-2.north east)
    -- (pic-2-2.south east) -- (pic-2-2.south west) --
    cycle;

\node[red] at (pic-1-1) {1};
\node[red] at (pic-2-2) {2};

\draw (a-1) circle[radius=5pt];
\draw (a-2) circle[radius=5pt];

\draw (pic-1-1) -- (pic-2-2);
\end{tikzpicture}
\end{document}
```
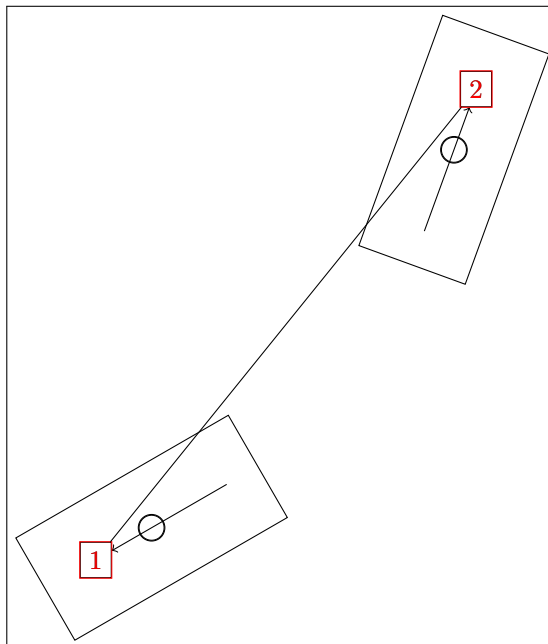
This produces:

5.2   Code Listings

If the `listings` package has been loaded then issuing

    \usetikzmarklibrary{listings}

will load in some code to add marks to `lstlisting` environments. This code places a mark at three places on a line of code in a `listings` environment. The marks are placed at the start of the line, the first non-whitespace character, and the end of the line (if the line is blank the latter two are not placed). (This has not been extensively tested, it works by adding code to various "hooks" that are made available by the `listings` package; it is quite possible that the hooks chosen are both wrong and insufficient to cover all desired cases.)

These are inspired by questions such as Marking lines in listings and Macros for code annotations.

In more detail, the `listings` library places lots of marks around the code. The marks are:

- `line-<name>-<number>-start` at the start of each line.

- `line-<name>-<number>-end` at the end of each line.

- `line-<name>-<number>-first` at the first non-space character of the line (assuming it exists).

The line numbers *should* match up with the line numbers in the code in that any initial offset is also applied.

Not every mark is available on every line. If a line is blank, in particular, it will only have a start mark. The following example shows this, where the red dots are the start, the blue are end, and the green are first.

```
\begin{tikzpicture}[remember picture]
\foreach \k in {0,...,7} {
\iftikzmark{line-code-\k-start}{\fill[red,overlay] (pic
    cs:line-code-\k-start)
    circle[radius=4pt];}{\message{No start for \k}}
\iftikzmark{line-code-\k-end}{\fill[blue,overlay] (pic
    cs:line-code-\k-end)
    circle[radius=2pt];}{\message{No end for \k}}
\iftikzmark{line-code-\k-first}{\fill[green,overlay]
    (pic cs:line-code-\k-first)
    circle[radius=2pt];}{\message{No first for \k}}
}
\draw[->,overlay] (0,0) -- (pic cs:line-code-5-first);
\draw[->,overlay] (0,0) -- (pic cs:line-code-5-start);
\draw[->,overlay] (0,0) -- (pic cs:line-code-5-end);
\node[above] at (0,0) {Line 5};
\end{tikzpicture}

\begin{lstlisting}[language=c,name=code,numbers=left]
  #include <stdio.h>

  int main(void)
  {
      printf("hello, world\n");
      return 0;
  }
\end{lstlisting}
```


This example puts a fancy node behind certain lines of the code, computing the necessary extents.

```
\balloon{comment}{more code}{3}{3}
\balloon{comment}{more code}{7}{8}
\begin{lstlisting}[language=c,name=more
    code,numbers=left,firstnumber=3]
  #include <stdio.h>

  int main(void)
  {
      printf("hello, world\n");
      return 0;
  }
\end{lstlisting}
```

```
3  #include <stdio.h>
4
5  int main(void)
6  {
7      printf("hello, world\n");
8      return 0;
9  }
```

6 Acknowledgements

The `\tikzmark` macro has been used and abused by many users of TeX-SX. Of particular note (but in no particular order) are Peter Grill, Gonzalo Medina, Claudio Fiandrino, percusse, and marmot. I would also like to mention David Carlisle whose knowledge of TikZ continues to astound us all.

7 Implementation

7.1 Main Code

The `save nodes` code uses LaTeX3.

```
1 \RequirePackage{expl3, l3keys2e, xparse}

2 \tikzset{%
3   remember picture with id/.style={%
4     remember picture,
5     overlay,
6     save picture id=#1,
7   },
```

Not totally happy with using `every picture` here as it's too easily overwritten by the user. Maybe it would be better to patch `endtikzpicture` directly.

```
8   every picture/.append style={%
```

```
9     execute at end picture={%
10      \ifpgfrememberpicturepositiononpage%
11      \edef\pgf@temp{%
12        \noexpand\write\noexpand\pgfutil@auxout{%
13          \string\savepicturepage%
14          {\pgfpictureid}{\noexpand\arabic{page}}%
15        }%
16      }%
17      \pgf@temp
18      \fi%
19    },
20  },
```

The positions are already recorded in the aux file, all we really need to do is provide them with better names.

```
21  save picture id/.code={%
22      \protected@write\pgfutil@auxout{}{%
23      \string\savepointas%
24      {\tikzmark@pp@name{#1}}{\pgfpictureid}{0pt}{0pt}}%
25  },
```

Provides a way to test if a picture has already been saved (in particular, can avoid errors on first runs)

```
26  if picture id/.code args={#1#2#3}{%
27    \@ifundefined{save@pt@\tikzmark@pp@name{#1}}{%
28      \pgfkeysalso{#3}%
29    }{
30      \pgfkeysalso{#2}%
31    }
32  },
```

Page handling

```
33  next page/.is choice,
34  next page vector/.initial={\pgfqpoint{0pt}{0pt}},
35  next page/below/.style={%
36    next page vector={\pgfqpoint{0pt}{-\the\paperheight}}%
37  },
38  next page/above/.style={%
39    next page vector={\pgfqpoint{0pt}{\the\paperheight}}%
40  },
41  next page/left/.style={%
42    next page vector={\pgfqpoint{-\the\paperwidth}{0pt}}%
43  },
44  next page/right/.style={%
45    next page vector={\pgfqpoint{\the\paperwidth}{0pt}}%
46  },
47  next page/ignore/.style={%
48    next page vector={\pgfqpoint{0pt}{0pt}}%
49  },
```

Prefix and suffix for tikzmark names, shamelessly borrowed from the main tikz code

```
50    tikzmark prefix/.initial=,%
51    tikzmark suffix/.initial=,%
52 }
```

\tikzmark@pp@name

```
53 \def\tikzmark@pp@name#1{%
54   \csname pgfk@/tikz/tikzmark prefix\endcsname%
55   #1%
56   \csname pgfk@/tikz/tikzmark suffix\endcsname%
57 }%
```

\savepointas   This is what gets written to the aux file.

```
58 \def\savepointas#1#2#3#4{%
59   \expandafter\gdef\csname save@pt@#1\endcsname{#2}%
60   \expandafter\gdef\csname save@pt@#1@offset\endcsname%
61   {\pgfqpoint{#3}{#4}}%
62 }
63 \def\savepicturepage#1#2{%
64   \expandafter\gdef\csname save@pg@#1\endcsname{#2}%
65 }
```

\tmk@labeldef   Auxiliary command for the coordinate system.

```
66 \def\tmk@labeldef#1,#2\@nil{%
67   \edef\tmk@label{\tikzmark@pp@name{#1}}%
68   \def\tmk@def{#2}%
69 }
```

pic   This defines the new coordinate system.

```
70 \tikzdeclarecoordinatesystem{pic}{%
71   \pgfutil@in@,{#1}%
72   \ifpgfutil@in@%
73     \tmk@labeldef#1\@nil
74   \else
75     \tmk@labeldef#1,(0pt,0pt)\@nil
76   \fi
77   \@ifundefined{save@pt@\tmk@label}{%
78     \tikz@scan@one@point\pgfutil@firstofone\tmk@def
79   }{%
80     \pgfsys@getposition{\csname save@pt@\tmk@label\endcsname}%
81     \save@orig@pic%
82     \pgfsys@getposition{\pgfpictureid}\save@this@pic%
83     \pgf@process{\pgfpointorigin\save@this@pic}%
84     \pgf@xa=\pgf@x
85     \pgf@ya=\pgf@y
86     \pgf@process{\pgfpointorigin\save@orig@pic}%
87     \advance\pgf@x by -\pgf@xa
88     \advance\pgf@y by -\pgf@ya
89     \pgf@xa=\pgf@x
90     \pgf@ya=\pgf@y
```

```
91      \pgf@process%
92      {\pgfpointorigin\csname save@pt@\tmk@label @offset\endcsname}%
93      \advance\pgf@xa by \pgf@x
94      \advance\pgf@ya by \pgf@y
95        \@ifundefined{save@pg@\csname save@pt@\tmk@label\endcsname}{}{%
96          \@ifundefined{save@pg@\pgfpictureid}{}{%
97            \pgfkeysvalueof{/tikz/next page vector}%
98          \advance \pgf@xa by %
99          \csname save@pg@\csname save@pt@\tmk@label\endcsname\endcsname\pgf@x\relax
100         \advance \pgf@ya by %
101         \csname save@pg@\csname save@pt@\tmk@label\endcsname\endcsname\pgf@y\relax
102         \advance \pgf@xa by %
103         -\csname save@pg@\pgfpictureid\endcsname\pgf@x\relax
104         \advance \pgf@ya by %
105         -\csname save@pg@\pgfpictureid\endcsname\pgf@y\relax
106         }%
107       }%
108       \pgf@x=\pgf@xa
109       \pgf@y=\pgf@ya
110     \pgftransforminvert
111     \pgf@pos@transform{\pgf@x}{\pgf@y}%
112     }%
113   }
```

\tikzmark   The active/non-active semi-colon is proving somewhat hazardous to \tikzmark
             (see Tikzmark and french seem to conflict and Clash between tikzmark, babel
             package (french) and babel tikzlibrary) so \tikzmark now uses the brace-delimited
             version of the \tikz command.

             This version is for when we're outside a tikzpicture environment

```
114 \newcommand\tikzmark@outside[2][]{%
115 \tikz[remember picture with id=#2]{#1}%
116 }
```

This is for when we're inside a tikzpicture environment

```
117 \def\tikzmark@inside#1#2{%
118   \tikzset{remember picture}%
119   \tikz@scan@one@point\pgfutil@firstofone#2\relax
120   \protected@write\pgfutil@auxout{}{%
121     \string\savepointas{\tikzmark@pp@name{#1}}{\pgfpictureid}{\the\pgf@x}{\the\pgf@y}}%
122 }
```

And finally, the ultimate invoker:

```
123 \def\tikzmark{%
124   \ifx\pgfpictureid\@undefined
125   \let\tikzmark@next=\tikzmark@outside
126   \else
127   \let\tikzmark@next=\tikzmark@inside
128   \fi
129   \tikzmark@next%
130 }
```

```
131 \newcommand\pgfmark[1]{%
132    \bgroup
133    \global\advance\pgf@picture@serial@count by1\relax%
134    \edef\pgfpictureid{pgfid\the\pgf@picture@serial@count}%
135      \pgfsys@markposition{\pgfpictureid}%
136    \edef\pgf@temp{%
137      \noexpand\write\noexpand\pgfutil@auxout{%
138        \string\savepicturepage{\pgfpictureid}{\noexpand\arabic{page}}}}%
139    \pgf@temp
140    \protected@write\pgfutil@auxout{}{%
141      \string\savepointas{\tikzmark@pp@name{#1}}{\pgfpictureid}{0pt}{0pt}}%
142    \egroup
143  }
```

If the beamer class is used, make the commands overlay aware.

```
144 \@ifclassloaded{beamer}{
145   \renewcommand<>{\tikzmark@outside}[2][]{%
146     \only#3{\beameroriginal{\tikzmark@outside}[{#1}]{#2}}%
147   }
148   \renewcommand<>{\tikzmark@inside}[2]{%
149     \only#3{\beameroriginal{\tikzmark@inside}{#1}{#2}}%
150   }
151 }{}
```

```
152 \@ifclassloaded{beamer}{
153   \renewcommand<>{\pgfmark}[1]{\only#2{\beameroriginal{\pgfmark}{#1}}}
154 }{}
```

If beamer is loaded, add a suffix based on the frame number

```
155 \@ifclassloaded{beamer}{
156   \tikzset{
157     tikzmark suffix=-\the\beamer@slideinframe
158   }
159 }{}
```

```
160 \newcommand\iftikzmark[3]{%
161    \@ifundefined{save@pt@\tikzmark@pp@name{#1}}{%
162      #3%
163    }{%
164      #2%
165    }%
166  }%
```

`\subnode`   Note: much of this code was inevitably adapted from the node defining code in
the TikZ/PGF sources.

```
167 \newcommand\subnode[3][]{%
168   \begingroup
169   \pgfmark{#2}%
170   \setbox\pgfnodeparttextbox=\hbox\bgroup #3\egroup
171   \def\tikz@shape{rectangle}%
172   \def\tikz@anchor{center}%
173   \def\tikz@fig@name{#2}%
174   \tikzset{every subnode/.try,#1}%
175   \pgfpointorigin
176   \tikz@scan@one@point\pgfutil@firstofone(pic cs:#2)\relax
177   \advance\pgf@x by .5\wd\pgfnodeparttextbox
178   \advance\pgf@y by .5\ht\pgfnodeparttextbox
179   \advance\pgf@y by -.5\dp\pgfnodeparttextbox
180   \pgftransformshift{}%
181   \setbox\@tempboxa=\hbox\bgroup
182   \pgfutil@ifundefined{pgf@sh@s@\tikz@shape}%
183   {\PackageError{pgf}{Unknown shape ''\tikz@shape''}{}}%
184   {%
185     {%
186       \let\pgf@sh@savedmacros=\pgfutil@empty% MW
187       \let\pgf@sh@savedpoints=\pgfutil@empty%
188       \def\pgf@sm@shape@name{\tikz@shape}% CJ % TT added prefix!
189       \csname pgf@sh@s@\tikz@shape\endcsname%
190       \pgf@sh@savedpoints%
191       \pgf@sh@savedmacros% MW
192       \pgftransformshift{%
193         \pgf@sh@reanchor{\tikz@shape}{\tikz@anchor}%
194         \pgf@x=-\pgf@x%
195         \pgf@y=-\pgf@y%
196       }%
197       \expandafter\pgfsavepgf@process\csname pgf@sh@sa@\tikz@fig@name\endcsname{%
198        \pgf@sh@reanchor{\tikz@shape}{\tikz@anchor}% FIXME : this is double work!
199       }%
200       % Save the saved points and the transformation matrix
201       \edef\pgf@node@name{\tikz@fig@name}%
202       \ifx\pgf@node@name\pgfutil@empty%
203       \else%
204         \expandafter\xdef\csname pgf@sh@ns@\pgf@node@name\endcsname{\tikz@shape}%
205       \edef\pgf@sh@@temp{%
206         \noexpand\gdef\expandafter\noexpand\csname pgf@sh@np@\pgf@node@name\endcsname}%
207       \expandafter\pgf@sh@@temp\expandafter{%
208         \pgf@sh@savedpoints}%
209       \edef\pgf@sh@@temp{%
210         \noexpand\gdef\expandafter\noexpand\csname pgf@sh@ma@\pgf@node@name\endcsname}% MW
211         \expandafter\pgf@sh@@temp\expandafter{\pgf@sh@savedmacros}% MW
212       \pgfgettransform\pgf@temp
213         \expandafter\xdef\csname pgf@sh@nt@\pgf@node@name\endcsname{\pgf@temp}%
```

```
214        \expandafter\xdef\csname pgf@sh@pi@\pgf@node@name\endcsname{\pgfpictureid}%
215      \fi%
216    }%
217  }%
218  \egroup
219  \box\pgfnodeparttextbox
220  \endgroup
221 }
```

\tikzmarknode   The \tikzmark macro has changed considerably since its first inception, but there
                 does still seem to be a use for the original version which put stuff inside a node.
                 This command reintroduces that command.

                 It does its best to work inside a math environment by a sneaky trick involving
                 \mathchoice: the remember picture key means that only the picture id of the
                 typeset box is saved to the aux file. So comparing the possible picture ids of the
                 four options with the one read from the aux file, we can figure out which box was
                 actually used.

```
222 \def\tikzmarknode@#1#2#3{%
223   \tikz[%
224     remember picture,
225     baseline=(#2.base),
226     save picture id={#2},
227     every tikzmarknode picture/.try
228   ] {
229     \node[
230       anchor=base,
231       inner sep=0pt,
232       name={#2},
233       node contents={#3},
234       every tikzmarknode/.try,
235       #1
236   ]}%
237 }
238
239 \newcommand\tikzmarknode[3][]{%
240   \ifmmode
241   \mathchoice{%
242     \tikzmarknode@{#1}{#2-d}{\(\displaystyle #3\)}%
243   }{%
244     \tikzmarknode@{#1}{#2-t}{\(\textstyle #3\)}%
245   }{%
246     \tikzmarknode@{#1}{#2-s}{\(\scriptstyle #3\)}%
247   }{%
248     \tikzmarknode@{#1}{#2-ss}{\(\scriptscriptstyle #3\)}%
249   }%
250   \let\pgf@nodecallback\pgfutil@gobble
251   \def\tzmk@prfx{pgf@sys@pdf@mark@pos@pgfid}%
252   \edef\tzmk@pic{\tzmk@prfx\the\pgf@picture@serial@count}
253   \expandafter\ifx\csname\tzmk@pic\endcsname\relax
```

```
254   \edef\tzmk@pic{\tzmk@prfx\the\numexpr\the\pgf@picture@serial@count-1\relax}%
255   \expandafter\ifx\csname\tzmk@pic\endcsname\relax
256 \edef\tzmk@pic{\tzmk@prfx\the\numexpr\the\pgf@picture@serial@count-2\relax}%
257   \expandafter\ifx\csname\tzmk@pic\endcsname\relax
258 \edef\tzmk@pic{\tzmk@prfx\the\numexpr\the\pgf@picture@serial@count-3\relax}%
259   \expandafter\ifx\csname\tzmk@pic\endcsname\relax
260     \pgfnodealias{#2}{#2-t}%  %% Fallback
261   \else
262     \pgfnodealias{#2}{#2-d}%
263   \fi
264   \else
265     \pgfnodealias{#2}{#2-t}%
266   \fi
267   \else
268     \pgfnodealias{#2}{#2-s}%
269   \fi
270   \else
271     \pgfnodealias{#2}{#2-ss}%
272   \fi
273   \else
274     \tikzmarknode@{#1}{#2}{#3}%
275   \fi
276 }
```

\usetikzmarklibrary

```
277 \def\usetikzmarklibrary{%
278   \pgfutil@ifnextchar[{\use@tikzmarklibrary}{\use@@tikzmarklibrary}%
279   }%}}
280 \def\use@tikzmarklibrary[#1]{\use@@tikzmarklibrary{#1}}
281 \def\use@@tikzmarklibrary#1{%
282   \edef\pgf@list{#1}%
283   \pgfutil@for\pgf@temp:=\pgf@list\do{%
284     \expandafter\pgfkeys@spdef\expandafter\pgf@temp\expandafter{\pgf@temp}%
285     \ifx\pgf@temp\pgfutil@empty
286     \else
287       \expandafter\ifx\csname tikzmark@library@\pgf@temp @loaded\endcsname\relax%
288       \expandafter\global\expandafter\let%
289       \csname tikzmark@library@\pgf@temp @loaded\endcsname=\pgfutil@empty%
290       \expandafter\edef\csname tikzmark@library@#1@atcode\endcsname{\the\catcode`\@}
291       \expandafter\edef\csname tikzmark@library@#1@barcode\endcsname{\the\catcode`\|}
292       \catcode`\@=11
293       \catcode`\|=12
294       \pgfutil@InputIfFileExists{tikzmarklibrary\pgf@temp.code.tex}{}{
295         \PackageError{tikzmark}{I did not find the tikzmark extras library '\pgf@temp'.}{}
296       }%
297       \catcode`\@=\csname tikzmark@library@#1@atcode\endcsname
298       \catcode`\|=\csname tikzmark@library@#1@barcode\endcsname
299       \fi%
300     \fi
301   }%
```

```
302 }
303
```

The `save node` code is written in LaTeX3.

```
304 \ExplSyntaxOn
```

We save our information in a "property list", which is L3's version of an associative array or dictionary. They keys will give the ability to store several groups of nodes and restore them at will.

```
305 \prop_new:N \g__sn_prop
```

We'll need a couple of spare token lists

```
306 \tl_new:N \l__sn_tmpa_tl
307 \tl_new:N \l__sn_tmpb_tl
```

Another useful token list

```
308 \tl_new:N \l__open_bracket_tl
309 \tl_set:Nn  \l__open_bracket_tl {[} %]
```

This token list is used for our current node group name

```
310 \tl_new:N \l__sn_group_tl
```

We store up the nodes in a list and save them at the end of a given tikzpicture.

```
311 \clist_new:N \l__sn_nodes_clist
```

This boolean is for whether we save to a file or not.

```
312 \bool_new:N \l__sn_file_bool
```

Dimensions and token lists for shifting

```
313 \dim_new:N \l__sn_x_dim
314 \dim_new:N \l__sn_y_dim
315 \dim_new:N \l__sn_xa_dim
316 \dim_new:N \l__sn_ya_dim
317 \tl_new:N \l__sn_centre_tl
318
319 \tl_new:N \l__sn_transformation_tl
320 \tl_set:Nn \l__sn_transformation_tl {{1}{0}{0}{1}{0pt}{0pt}}
```

Set up a stream for saving the nodes data to a file

```
321 \iow_new:N \g__sn_stream
322 \bool_new:N \g__sn_stream_bool
323 \tl_new:N \g__sn_filename_tl
324 \tl_set:Nx \g__sn_filename_tl {\c_sys_jobname_str}
325
326 \cs_new_nopar:Npn \sn_open_stream:
327 {
328   \bool_if:NF \g__sn_stream_bool
329   {
330     \iow_open:Nn \g__sn_stream {\tl_use:N \g__sn_filename_tl .nodes}
331     \bool_gset_true:N \g__sn_stream_bool
332   }
333 }
```

```
334
335 \AtEndDocument
336 {
337   \ExplSyntaxOn
338   \bool_if:NT \g__sn_stream_bool
339   {
340     \iow_close:N \g__sn_stream
341   }
342   \ExplSyntaxOff
343 }
```

LaTeX3 wrappers around some PGF functions (to avoid @-catcode issues)

```
344 \makeatletter
345 \cs_set_eq:NN \tikz_set_node_name:n \tikz@pp@name
346 \cs_set_eq:NN \tikz_fig_must_be_named: \tikz@fig@mustbenamed
347
348 \cs_new_nopar:Npn \tikz_scan_point:n #1
349 {
350   \tikz@scan@one@point\pgfutil@firstofone#1\relax
351 }
352
353 \cs_new_nopar:Npn \tikz_scan_point:NNn #1#2#3
354 {
355   \tikz@scan@one@point\pgfutil@firstofone#3\relax
356   \dim_set_eq:NN #1 \pgf@x
357   \dim_set_eq:NN #2 \pgf@y
358 }
359
360 \makeatother
361 \cs_generate_variant:Nn \tikz_scan_point:n {V}
362 \cs_generate_variant:Nn \tikz_scan_point:NNn {NNV}
```

\save_nodes:Nn   This is the command that actually does the work. It constructs a token list which contains the code that will restore the node data when invoked. The two arguments are the token list to store this in and a comma separated list of the node names to be saved.

```
363 \cs_new_nopar:Npn \save_nodes:Nn #1#2
364 {
```

Clear our token lists

```
365   \tl_clear:N \l__sn_tmpa_tl
```

Set the centre of the picture

```
366   \tikz_scan_point:NNn \l__sn_x_dim \l__sn_y_dim {(current~ bounding~ box.center)}
367   \dim_set:Nn \l__sn_x_dim {-\l__sn_x_dim}
368   \dim_set:Nn \l__sn_y_dim {-\l__sn_y_dim}
369   \tl_set:Nx \l__sn_centre_tl {
370     {1}{0}{0}{1}{\dim_use:N \l__sn_x_dim}{\dim_use:N \l__sn_y_dim}
371   }
```

Iterate over the list of node names

```
372   \clist_map_inline:nn {#2}
373   {
```

Test to see if the node has been defined

```
374       \tl_if_exist:cT {pgf@sh@ns@##1}
375       {
```

The node information is stored in a series of macros of the form `\pgf@sh@XX@nodename` where XX is one of the following.

```
376         \clist_map_inline:nn {ns,np,ma,pi}
377         {
```

Our token list will look like:

    `\tl_set:cn {pgf@sh@XX@nodename}` <current contents of that macro>

    This will restore `\pgf@sh@XX@nodename` to its current value when this list is invoked.

    This part puts the `\tl_set:cn {pgf@sh@XX@nodename}` in place

```
378           \tl_put_right:Nn \l__sn_tmpa_tl
379           {
380             \tl_gset:cn {pgf@sh@####1@ \tikz_set_node_name:n{##1} }
381           }
```

Now we put the current contents in place. We're doing this in an expansive context to get at the contents. The `\exp_not:v` part takes the current value of `\pgf@sh@XX@nodename` and puts it in place, preventing further expansion.

```
382           \tl_if_exist:cTF {pgf@sh@####1@##1}
383           {
384             \tl_put_right:Nx \l__sn_tmpa_tl {
385               {\exp_not:v {pgf@sh@####1@ \tikz_set_node_name:n {##1}}}
386           }
387           }
388           {
389             \tl_put_right:Nx \l__sn_tmpa_tl {{}}
390           }
391         }
392         \tl_put_right:Nn \l__sn_tmpa_tl
393         {
394           \tl_gset:cn {pgf@sh@nt@ \tikz_set_node_name:n{##1} }
395         }
396         \compose_transformations:NVv \l__sn_tmpb_tl \l__sn_centre_tl {pgf@sh@nt@##1}
397         \tl_put_right:Nx \l__sn_tmpa_tl {{\exp_not:V \l__sn_tmpb_tl}}
398         \tl_put_right:Nn \l__sn_tmpa_tl {
399           \transform_node:Nn \l__sn_transformation_tl {
400             \tikz_set_node_name:n{##1}
401           }
402         }
403     }
404   }
```

Once we've assembled our token list, we store it in the given token list

```
405   \tl_set_eq:NN #1 \l__sn_tmpa_tl
406 }
```

`\save_nodes_to_list:nn`   Save the nodes to a list, given a key

```
407 \cs_new_nopar:Npn \save_nodes_to_list:nn #1#2
408 {
409   \save_nodes:Nn \l__sn_tmpa_tl {#2}
410   \prop_gput:NnV \g__sn_prop {#1} \l__sn_tmpa_tl
411 }
```

`\save_nodes_to_file:n`   Save the nodes to a file

```
412 \cs_new_nopar:Npn \save_nodes_to_file:n #1
413 {
414   \save_nodes:Nn \l__sn_tmpa_tl {#1}
```

Save the token list to the aux file so that on reading the aux file, we restore the node definitions

```
415   \sn_open_stream:
416   \iow_now:Nx \g__sn_stream
417   {
418     \iow_newline:
419     \exp_not:V \l__sn_tmpa_tl
420     \iow_newline:
421   }
422 }

423 \cs_generate_variant:Nn \save_nodes_to_list:nn {VV}
424 \cs_generate_variant:Nn \save_nodes_to_file:n {V}
```

`\restore_nodes_from_list:n`

```
425 \cs_new_nopar:Npn \restore_nodes_from_list:n #1
426 {
```

Restoring nodes is simple: look in the property list for the key and if it exists, invoke the macro stored there.

```
427   \prop_get:NnNT \g__sn_prop {#1} \l__sn_tmpa_tl
428   {
429     \tl_use:N \l__sn_tmpa_tl
430   }
431 }
```

`\restore_nodes_from_file:n`

```
432 \cs_new_nopar:Npn \restore_nodes_from_file:n #1
433 {
434   \file_if_exist:nT {#1.nodes}
435   {
436     \ExplSyntaxOn
437     \file_input:n {#1.nodes}
438     \ExplSyntaxOff
439   }
440 }
```

`\compose_transformations:Nnn` Compose PGF transformations `#2 * #3`, storing the result in `#1`

I think the PGF Manual might be incorrect. It implies that the matrix is stored row-major, but experimentation implies column-major.

That is, `{a}{b}{c}{d}{s}{t}` is:

$$\begin{bmatrix} a & c \\ b & d \end{bmatrix}$$

```
441 \cs_new_nopar:Npn \compose_transformations:Nnn #1#2#3
442 {
443   \tl_gset:Nx #1
444   {
445     {\fp_eval:n {
446         \tl_item:nn {#2} {1}
447         * \tl_item:nn {#3} {1}
448         +
449         \tl_item:nn {#2} {3}
450         * \tl_item:nn {#3} {2}
451     }
452   }
453     {\fp_eval:n {
454         \tl_item:nn {#2} {2}
455         * \tl_item:nn {#3} {1}
456         +
457         \tl_item:nn {#2} {4}
458         * \tl_item:nn {#3} {2}
459     }
460   }
461     {\fp_eval:n {
462         \tl_item:nn {#2} {1}
463         * \tl_item:nn {#3} {3}
464         +
465         \tl_item:nn {#2} {3}
466         * \tl_item:nn {#3} {4}
467     }
468   }
469     {\fp_eval:n {
470         \tl_item:nn {#2} {2}
471         * \tl_item:nn {#3} {3}
472         +
473         \tl_item:nn {#2} {4}
474         * \tl_item:nn {#3} {4}
475     }
476   }
477     {\fp_to_dim:n {
478         \tl_item:nn {#2} {1}
479         * \tl_item:nn {#3} {5}
480         +
481         \tl_item:nn {#2} {3}
482         * \tl_item:nn {#3} {6}
```

```
483         +
484           \tl_item:nn {#2} {5}
485       }
486     }
487     {\fp_to_dim:n {
488           \tl_item:nn {#2} {2}
489         * \tl_item:nn {#3} {5}
490         +
491           \tl_item:nn {#2} {4}
492         * \tl_item:nn {#3} {6}
493         +
494           \tl_item:nn {#2} {6}
495       }
496     }
497   }
498 }

499 \cs_generate_variant:Nn \compose_transformations:Nnn {cVv,NVv,NVn,NvV,NnV}
```

\transform_node:Nn

```
500 \cs_new_nopar:Npn \transform_node:Nn #1#2
501 {
502   \compose_transformations:cVv {pgf@sh@nt@#2} #1 {pgf@sh@nt@#2}
503 }
```

\set_transform_from_node:n

```
504 \cs_new_nopar:Npn \set_transform_from_node:n #1
505 {
506   \tl_set_eq:Nc \l__sn_transformation_tl {pgf@sh@nt@#1}
507   \tikz_scan_point:NNn \l__sn_x_dim \l__sn_y_dim {(#1.center)}
508
509   \dim_set:Nn \l__sn_x_dim {\l__sn_x_dim - \tl_item:cn {pgf@sh@nt@#1}{5}}
510   \dim_set:Nn \l__sn_y_dim {\l__sn_y_dim - \tl_item:cn {pgf@sh@nt@#1}{6}}
511
512   \compose_transformations:NnV  \l__sn_transformation_tl {
513     {1}{0}{0}{1}{\dim_use:N \l__sn_x_dim}{\dim_use:N \l__sn_y_dim}
514   } \l__sn_transformation_tl
515 }

516 \cs_generate_variant:Nn \set_transform_from_node:n {v}
```

Set the TikZ keys for access to the above commands.

```
517 \tikzset{
518   set~ saved~ nodes~ file~ name/.code={
519     \tl_gset:Nx \g__sn_filename_tl {#1}
520   },
521   transform~ saved~ nodes/.code={
522     \set_transform_from_node:v {tikz@last@fig@name}
523   },
524   set~ node~ group/.code={
```

```
525    \tl_set:Nn \l__sn_group_tl {#1}
526    \pgfkeysalso{
527      execute~ at~ end~ scope={
528        \maybe_save_nodes:
529      }
530    }
531  },
```

Are we saving to a file?

```
532  save~ nodes~ to~ file/.code={
533    \tl_if_eq:nnTF {#1}{false}
534    {
535      \bool_set_false:N \l__sn_file_bool
536    }
537    {
538      \bool_set_true:N \l__sn_file_bool
539    }
540    \pgfkeysalso{
541      execute~ at~ end~ scope={
542        \maybe_save_nodes:
543      }
544    }
545  },
```

Append current node to the list of nodes to be saved

```
546  save~ node/.code={
547    \tikz_fig_must_be_named:
548    \pgfkeysalso{append~ after~ command={
549        \pgfextra{
550          \clist_gput_right:Nv \l__sn_nodes_clist {tikz@last@fig@name}
551        }
552      }
553    }
554  },
```

Restore nodes from file

```
555  restore~ nodes~ from~ file/.code={
556    \tikz_fig_must_be_named:
557    \pgfkeysalso{append~ after~ command={
558        \pgfextra{
559          \scope
560          \split_argument:NNn \tikzset \restore_nodes_from_file:n {#1}
561          \endscope
562        }
563      }
564    }
565  },
```

Restore nodes from list

```
566  restore~ nodes~ from~ list/.code={
567    \tikz_fig_must_be_named:
```

```
568    \pgfkeysalso{append~ after~ command={
569        \pgfextra{
570            \scope
571            \split_argument:NNn \tikzset \restore_nodes_from_list:n {#1}
572            \endscope
573        }
574      }
575    }
576  }
577 }
578 \cs_generate_variant:Nn \clist_gput_right:Nn {Nv}
```

\split_argument:NNn

```
579 \cs_new_nopar:Npn \split_argument:NNn #1#2#3
580 {
581   \tl_set:Nx \l__sn_tmpa_tl {\tl_head:n {#3}}
582   \tl_if_eq:NNTF \l__sn_tmpa_tl \l__open_bracket_tl
583   {
584     \split_argument_aux:NNp #1#2#3
585   }
586   {
587     #2 {#3}
588   }
589 }
```

\split_argument_aux:NNp

```
590 \cs_new_nopar:Npn \split_argument_aux:NNp #1#2[#3]#4
591 {
592   #1 {#3}
593   #2 {#4}
594 }
```

\maybe_save_nodes:

```
595 \cs_new_nopar:Npn \maybe_save_nodes:
596 {
597   \clist_if_empty:NF \l__sn_nodes_clist
598   {
599     \bool_if:NTF \l__sn_file_bool
600     {
601       \save_nodes_to_file:V \l__sn_nodes_clist
602     }
603     {
604       \tl_if_empty:NF \l__sn_group_tl
605       {
606         \save_nodes_to_list:VV \l__sn_group_tl \l__sn_nodes_clist
607       }
608     }
609     \clist_gclear:N \l__sn_nodes_clist
610   }
611 }
```

612 `\ExplSyntaxOff`

7.2 Listings

From `http://tex.stackexchange.com/q/79762/86`

613 `\@ifpackageloaded{listings}{%`

`\iflst@linemark` A conditional to help with placing the mark at the first non-whitespace character.

614   `\newif\iflst@linemark`

`EveryLine` This hook places the mark at the start of the line.

615 `\lst@AddToHook{EveryLine}{%`
616   `\begingroup`
617   `\advance\c@lstnumber by 1\relax`
618   `\pgfmark{line-\lst@name-\the\c@lstnumber-start}%`
619   `\endgroup`
620 `}`

`EOL` This hook places the mark at the end of the line and resets the conditional for placing the first mark.

621 `\lst@AddToHook{EOL}{\pgfmark{line-\lst@name-\the\c@lstnumber-end}%`
622 `\global\lst@linemarktrue`
623 `}`

`OutputBox` Experimenting shows that this is the right place to set the mark at the first non-whitespace character. But we only want to do this once per line.

624 `\lst@AddToHook{OutputBox}{%`
625   `\iflst@linemark`
626   `\pgfmark{line-\lst@name-\the\c@lstnumber-first}%`
627   `\global\lst@linemarkfalse`
628   `\fi`
629 `}`

`\tikzmk@lst@fnum` An auxiliary macro to figure out if the `firstnumber` key was set. If so, it has the form `<number>\relax`. If not, it expands to a single token.

630 `\def\tkzmk@lst@fnum#1\relax#2\@STOP{%`
631   `\def\@test{#2}%`
632   `\ifx\@test\@empty`
633   `\def\tkzmk@lst@start{0}%`
634   `\else`
635   `\@tempcnta=#1\relax`
636   `\advance\@tempcnta by -1\relax`
637   `\def\tkzmk@lst@start{\the\@tempcnta}%`
638   `\fi`
639 `}`

`Init` Adds a mark at the start of the listings environment.

640 `\lst@AddToHook{Init}{%`

```
641    \expandafter\tkzmk@lst@fnum\lst@firstnumber\relax\@STOP
642    \pgfmark{line-\lst@name-\tkzmk@lst@start-start}%
643 }

644 }{%
645    \PackageError{tikzmark listings}{The listings package has not been loaded.}{}
646 }
```