tile-graphic: Break a graphic into tiles

D. P. Story
Email: `dpstory@acrotex.net`

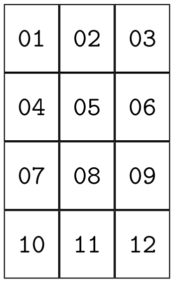
processed May 28, 2020

Contents

```
⟨*package⟩
```

1 Documentation

1.1 A brief description



01	02	03
04	05	06
07	08	09
10	11	12

Tiled graphic
(4×3)

The document author opens the `create-tg.tex` file and specifies three arguments of the `\setTileParams` command: a number n that corresponds to the number of rows; a number m that represents the number of columns; and the path to a graphic. Compiling `create-tg.tex`, when the option `wrttofiles` is specified, results in the specified graphic being broken down into a series of $n \times m$ *tiled* graphics. The order of the creation of the tiles is top to bottom, left to right: The first row is the top most row, and the graphic is tiled across the row, from left to right, the next row, is the one just below the top row, and the graphic is tiled across that row, from left to right, and so on. Refer to the crude diagram to the left.

> **Warning:** This package uses the shellesc package, which requires the `--shell-escape` switch. Use this package only if you trust the author of this package.

1.2 Applications

The tile-graphic package can be used to produce tiled graphics, which can be consumed by the dps and acrosort packages.

1.3 The `create-tg.tex` file

Because of the extensive use of `\ShellEscape`, `create-tg.tex` is similar to a BAT (batch) file. When you compile `create-tg.tex`, one result is `create-tg.pdf`; `create-tg.pdf` is a one page summary titled **Tile Graphic Report**. In addition to producing a report, there is the actual result the report refers to: the production of tiled graphic files (PDFs).

Tiled graphics are obtained by compiling `create-tg.tex`, after setting three parameters; using this package, therefore, is very simple. Below is the verbatim listing of the `create-tg.tex`, found in the `examples` folder.

```
    \documentclass{article}
(1) \usepackage[⟨options⟩]{tile-graphic}
(2) \setTileParams[⟨ig-opts⟩]{⟨nRows⟩}{⟨nCols⟩}{⟨path⟩}
    \begin{document}
(3) \tileTheGraphic
    \end\darg{document}
```

When `create-tg.tex` is compiled, the DVI (in the case of LaTeX) or the PDF (in all other cases) produces a single page document the title of which is "**Tile Graphic Report**." The file also produces, depending on the options used, various *separate* PDFs consisting of tiled graphics.

Discussion. We discuss each of the numbered lines slightly out of order.

`\setTileParams` (2) `\setTileParams[⟨ig-opts⟩]{⟨nRows⟩}{⟨nCols⟩}{⟨path⟩}`
The command and its arguments are placed in the preamble.

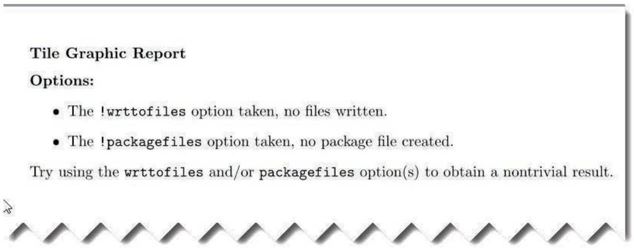
⟨*ig-opts*⟩ This optional argument is passed to the underlying optional argument of the `\includegraphics` command. Normally, there is no optional options passed.

⟨*nRows*⟩ This argument declares the number of rows you want to break the graphic into.

⟨*nCols*⟩ This argument declares the number of columns you want to break the graphic into.

⟨*path*⟩ The path to the graphic. The graphic is any file format supported by the PDF creator. For a `latex->dvips->`⟨`ps2pdf|distiller`⟩ workflow, the graphic should be an EPS file; in all other workflows, the graphic can be a PDF (or some other supported graphical format). For example, if the graphic is in the `graphics` folder of the source file, then ⟨*path*⟩ might read `graphics/mygraphic`, where `mygraphic.pdf` (for example) is in the `graphics` folder.

With no package options, compiling `create-tg.tex` produces a document a single page document with a message, seen in Figure 1.



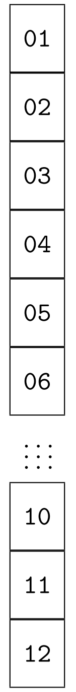
Tile Graphic Report

Options:

- The `!wrttofiles` option taken, no files written.
- The `!packagefiles` option taken, no package file created.

Try using the `wrttofiles` and/or `packagefiles` option(s) to obtain a nontrivial result.

Figure 1: No options



Pages (4 × 3)

(1) `\usepackage[`⟨*options*⟩`]{tile-graphic}`
For the package declaration, the ⟨*options*⟩ are described below.

`wrttofiles` When `create-tg.tex` is compiled with the `wrttofiles` option *only* and with a ⟨*path*⟩ argument of `graphics/mygraphic`, ⟨*nRows*⟩ × ⟨*nCols*⟩ individual tile PDF files named

`mygraphic_01.pdf`, `mygraphic_02.pdf`, ...

are created and placed in the `graphics` folder. If no subfolder is specified, files are placed in the source file's folder. The source file that is compiled (`create-tg.pdf`) contains the **Tile Graphic Report**.

`!wrttofiles` (Convenience option) This option reverts to the default, no separate tile PDFs are produced.

`packagefiles` When this option is specified, ⟨*nRows*⟩ × ⟨*nCols*⟩ pages are produced and "packaged" in a separate PDF document named ⟨*basename*⟩`_package.pdf`, where ⟨*basename*⟩ is the base name of the graphic. See the crude illustration to the left.

`!packagefiles` (Convenience option) Reverses the `packagefiles` option, as a result, no package file is created.

`pdfcreator=`⟨`pdflatex|lualatex|xelatex|ps2pdf|distiller`⟩
When the option `wrttofiles` or `packagefiles` is specified, compiling `create-tg.tex` produces one or more PDF files. These PDFs are produced by pdflatex, lualatex, xelatex, ps2pdf, or distiller by executing a series of `\ShellEscape` commands. The PDF creator application

used is determined by the value of the `pdfcreator` key. The default is `pdflatex`.

Using the `\ShellEscape` command is system dependent. The tile-graphic package is set up for the Windows OS; it uses del, copy, and move. Refer to Section 2.4 for more information for modifying these system commands.

tile-graphic also uses `\ShellEscape` to compile certain dynamically created TEX files. There are hooks to modify the compile commands that can be used at your discretion. Again, refer to Section 2.4 for more information.

`\tileTheGraphic` (3) `\tileTheGraphic` is placed in the body of the document. It is this command that does all the work, based on the parameters of `\setTileParam`.

Required packages. The following package are required: shellesc, web, graphicx, and multido.

Try it, you'll like it. The first thing to do, once the package is installed, is to try the compiling the file `create-tg.tex`. There are three graphics files, one at the top level of the `examples` folder, one in the `choo` subfilder, and one in the `postscript` subfolder. Try both graphics (only one graphic at a time, however). Try all possible combinations of the basic options `wrttofiles`, `!wrttofiles`, `packagefiles`, and `!packagefiles`; you can even try the various values of `pdfcreator`.

1.4 Description of the workflow

Be aware that there are two instances, or stages, of compiling:

Stage 1. Compiling `create-tg.tex`, usually, this is initiated in some LaTeX editor. At this stage, you can compile with latex.exe, pdflatex.exe, lualatex.exe, or xelatex.exe. When `\setTileParams` is expanded, it reads the graphic file, ⟨*path*⟩, using `\includegraphics` to determine its size. Therefore, the graphic must be one that is supported by the pdf creator at this point. For example if the graphic is a PDF, yet you use latex.exe to compile `create-tg.tex`, you get an error because latex.exe does not support PDF inclusion. `\setTileParams`

Stage 2. Compiling of one or more files to produce tiled graphics, this step is initiated by the `\ShellEscape` command. The application used to compile at this stage is determined by the `pdfcreator` key. When the `packagefiles` option is taken, the `\setTileParams` is expanded again and the graphic file, ⟨*path*⟩, is included to determine dimension of the graphic. Again, the graphic must be a format supported by the PDF creator as selected by the `pdfcreator` key, which may be different from the compiler of the first case above. `\setTileParams`

Additional thoughts on the workflow.

- `pdfcreator=`⟨`pdflatex|lualatex|xelatex`⟩

  This is the easiest and simplest case. Usually the PDF creators for the two stages of compiling are the same (refer to Stages 1 and 2, above). These are direct-to-PDF applications, the workflow is relatively quick, smooth, and automated. All AUX files are are deleted except for the AUX files of the main file `create-tg.tex`.

- `pdfcreator=`⟨`ps2pdf|distiller`⟩

  The graphic file format must be EPS for Stage 2; however, if you are not using latex.exe in Stage 1, the graphic must be one that is supported by the PDF creator used. Thus you may have two copies of the graphic of different formats. You're not going to do that, are you? Usually the PDF creators for the two stages of compiling are the same, that way you need not have two formats for the graphic.

  - `pdfcreator=`⟨`ps2pdf`⟩

    The workflow is `latex -> dvips -> ps2pdf`

  - `pdfcreator=`⟨`distiller`⟩

    The workflow is `latex -> dvips -> distiller`. There is a difference between using distiller and ps2pdf, in the latter case, the PS file can be a relative path; in the former case, it must be a full (or absolute) path, this is a slight complication. For this bad boy, we provide the `\fullPathToSource`, refer to Section 2.4 for more details.

`\fullPathToSource`

1.5 The Configuration File

`tg.cfg`

A configuration file, named `tg.cfg`, is input at the end of the package. You might make any customizations there, where you customize for the pdf creator you always use, or, if you use several, you can make customizations for each by using the `\ifcase` structure below.

```
\ifcase\tg@case\relax
  \def\tg@pdfcreator@app{pdflatex}
    ⟨customizations-for-pdflatex⟩
\or
  \def\tg@pdfcreator@app{lualatex}
    ⟨customizations-for-lualatex⟩
\or
  \def\tg@pdfcreator@app{xelatex}
    ⟨customizations-for-xelatex⟩
\or
  \def\tg@pdfcreator@app{ps2pdf}
    ⟨customizations-for-ps2pdf⟩
\or
  \def\tg@pdfcreator@app{acrodist}
```

```
        ⟨customizations-for-acrodist⟩
    \fi
```

`tg.cfg.txt` See the file `tg.cfg.txt` for this structure. Rename it to `tg.cfg` if found to be useful.

End of Documentation

2 The package code

Having finished the barest of documentation, we begin the markup of this package.

2.1 Options

We bring in the xkeyval package so we can gather our options using it's commands, rather than the default keyval commands.

```
\RequirePackage{xkeyval}
```

`wrttofiles` `!wrttofiles` When `wrttofiles` is specfied, ⟨*nRows*⟩ × ⟨*nCols*⟩ PDF files are created. When `!wrttofiles` is used, these files just mentions are not created.

```
\newif\if@wrttofiles\@wrttofilesfalse
\DeclareOptionX{wrttofiles}{\@wrttofilestrue}
\DeclareOptionX{!wrttofiles}{\@wrttofilesfalse}
```

`packagefiles` `!packagefiles` When specified, a single "package" PDF is created containing ⟨*nRos*⟩ × ⟨*nCols*⟩ pages of tiled graphics Likewise, the option `!packagefiles` reverses `packagefiles` to return to the default state, files are not packaged.

```
\newif\if@packagefiles \@packagefilestrue
\DeclareOptionX{packagefiles}{\@packagefilestrue}
\DeclareOptionX{!packagefiles}{\@packagefilesfalse}
```

`pdfcreator=⟨pdflatex|lualatex|xelatex|ps2pdf|distiller⟩`

```
\define@choicekey*+{tile-graphic.sty}{pdfcreator}[\val\nr]%
  {pdflatex,lualatex,xelatex,ps2pdf,distiller}{%
  \edef\tg@case{\nr}%
  \ifcase\nr\relax
    \def\tg@pdfcreator@app{pdflatex}\or
    \def\tg@pdfcreator@app{lualatex}\or
    \def\tg@pdfcreator@app{xelatex}\or
    \def\tg@pdfcreator@app{ps2pdf}\or
    \def\tg@pdfcreator@app{acrodist}\fi
}{\PackageWarning{tile-graphics}%
  {Bad choice for pdfcreator, permissible values\MessageBreak
  are pdflatex, lualatex, xelatex, ps2pdf, and\MessageBreak
  distiller.\MessageBreak Using the default pdflatex}}
\def\tg@pdfcreator@app{pdflatex}\def\tg@case{0}
```

Process the options

```
\ProcessOptionsX\relax
\edef\tg@restoreCats{%
  \catcode‘\noexpand\"=\the\catcode‘\"\relax
  \catcode‘\noexpand\,=\the\catcode‘\,\relax
  \catcode‘\noexpand\_=\the\catcode‘\_\relax
}
\@makeother\"\@makeother\,\@makeother\_
```

2.2 Required Packages

The web package is use to set the page dimensions, it also brings in hyperref. We use graphicx package to have access to `\includegraphics`. The `\multido` package is used to work across the face of the graphic by row, top to bottom, to clip off little bounding boxes of the graphic. shellesc is required for the `wrttofiles` option.

```
\RequirePackage{shellesc}
\RequirePackage{web}
\RequirePackage{graphicx}
\RequirePackage{multido}
```

2.3 Registers and other preliminaries

`\if@wrtorpkg`

We declare a new if, `\if@wrtorpkg`, which is true if either `\if@wrttofiles` or `\if@packagefiles` is true. This is a convenience to the coding.

```
\newif\if@wrtorpkg \@wrtorpkgfalse
\if@wrttofiles
  \@wrtorpkgtrue
\else
  \if@packagefiles
    \@wrtorpkgtrue
  \fi
\fi
\newif\iftg@direct \tg@directtrue
\newif\iftgfolder  \tgfolderfalse
```

`\iftg@direct`

`\iftg@direct` is false if `pdfcreator=`⟨*ps2pdf|distiller*⟩.

```
\ifnum\tg@case>\tw@\relax\tg@directfalse\fi
```

A few comments on the `\ifpassthruTG` switch. This package performs some tricks. Initially, `\ifpassthruTG` is true, and certain portions of the code are executed. When `create-tg.tex` is compiled with `\passthruTGtrue`, it writes to the current folder the file `package-data.cut` containing the single command `\passthruTGfalse`. During this compile, `create-tg.tex` makes a copy of itself named ⟨*basename*⟩`_package.tex`. It is this tile that is compiled by `\ShellEscape`, and when compiled, reads `package-data.cut`, since it now exists, and `\ifpassthruTG` is set to false, at which point a different set of code lines are executed.

```
\newif\ifpassthruTG \passthruTGtrue
```

Some miscellaneous lengths, boxes, and utility commands.

```
\newdimen\tg@dima
\newdimen\tg@dimb
\newbox\tg@box
\newcount\tg@Cnt
\def\tg@rmslash#1/{\def\tgInFolder{#1}}%
\let\tgInFolder\@empty
```

2.4 Some customization commands

`\sysdel` `\syscopy` `\sysmove`

System commands. These are `\sysdel`, `\syscopy`, and `\sysmove`. The default is to use Windows OS commands. Changes for UNIX/Linux or MacOS are easy enough by declaring these three commands in the preamble (or in `tg.cfg`), their arguments are the names of the corresponding system commands for deleting, copying, and moving files.

```
\def\sysdel#1{\def\tg@sysdel{#1\space}}
\sysdel{del}
\def\syscopy#1{\def\tg@syscopy{#1\space}}
\syscopy{copy}
\def\sysmove#1{\def\tg@sysmove{#1\space}}
\sysmove{move}
```

`\pkgappArgs`

Changing the switches in Stage 2. Here is a link to the Stage 2 reference. There are two TEX files compiled: (1) the package file, and (2) the individual tiles graphic files. We provide a hook to modify the command line switches of the `pdfcreator`. `\pkgappArgs` is the hook into the creation of the package file. Use `#1` to represent the `pdfcreator` application. For example, `\pkappArgs{#1 -halt-on-error}` produces a command line of

> pdflatex -halt-on-error –shell-escape ⟨*targeted-package-file.tex*⟩

For a `pdfcreator` of pdflatex. The `--shell-escape` is automatically included.

```
\def\pkgappArgs#1{\def\pkg@ppArgs##1{#1}}
\pkgappArgs{#1}
```

`\tileappArgs`

We also separately compile each of the separate tile graphics. `\tileappArgs` is similar to `\pkgappArgs`. If we declare `\tileappArgs{#1 -halt-on-error}`, then the command line becomes,

> pdflatex -halt-on-error ⟨*targeted-tile-graphic.tex*⟩

Here, `--shell-escape` is not automatically included as it is not needed for this step.

```
\def\tileappArgs#1{\def\tile@ppArgs##1{#1}}
\tileappArgs{#1}
```

Some unverified tricks. What if your system does not recognize pdflatex, lualatex, and so on; perhaps they are know by different names. You can use these hooks to fix that. For example,

`\pkgappArgs{⟨alt-pdflatex⟩.exe ⟨other-switches⟩}`

Or, perhaps the path to this application is not on the system paths, hence pdflatex is not recognized, in this case, try

`\pkgappArgs{⟨path-to⟩/⟨alt-pdflatex⟩.exe ⟨other-switches⟩}`

that "should" work.

`\latexappArgs`
`\dvipsappArgs`

When `pdfcreator=`⟨*ps2pdf|distiller*⟩, we provide two hooks for this workflow: `\latexappArgs` and `\dvipsappArgs`. These function similar to the ones described above; for example, `\dvipsappArgs{#1 -q*}` passes the `-q*` switch (quiet mode) to dvips. No separate commands for a package compile versus a tile compile, as seen above, are defined, though that could change.

```
\def\latexappArgs#1{\def\latex@ppArgs##1{#1}}
\latexappArgs{#1}
\def\dvipsappArgs#1{\def\dvips@ppArgs##1{#1}}
\dvipsappArgs{#1}
```

`\definePath{⟨\cmd⟩}{⟨path⟩}` A command taken from eforms that normalizes the argument before defining `\cmd`.

```
\providecommand{\definePath}[1]{\def\ef@ctrlName{#1}%
    \hyper@normalise\ef@definePath}
\def\ef@definePath#1{\expandafter\xdef\ef@ctrlName{#1}}
```

Support for `pdfcreator=distiller`.

`\fullPathToSource{⟨path⟩}` Set the full path to the source file (`create-tg.tex`). This command is only needed when `pdfcreator=distiller`.

```
\def\fullPathToSource{\definePath{\tg@fullPathToSource}}
\let\tg@fullPathToSource\@empty
```

After creation hooks. We provide additional hooks, the first is placed just after a tile file is created, and second one is placed just after the package file is created.

`\afterTileCreationHook{⟨\cmds⟩}` The ⟨*\cmds*⟩ can be any valid LaTeX commands, conceptually, the commands may contain `\ShellEscape` commands. The hook is placed just after a tile file is created as a PDF.

`\afterPkgCreationHook{⟨\cmds⟩}` Similar to `\afterTileCreationHook`, but for the creation of the tile package.

```
\def\afterTileCreationHook#1{\def\@fterTileCreationHook{#1}}
\let\@fterTileCreationHook\relax
```

The definition of `\afterPkgCreationHook`.

```
\def\afterPkgCreationHook#1{\def\@fterPkgCreationHook{#1}}
\let\@fterPkgCreationHook\relax
```

Examples. This example makes each *tile* PDF file into a tile EPS file. We use the utility executable pdftops, which may be available on your TEX system. In the preamble or above the `\tileTheGraphic` command, place the following commands.

```
\afterTileCreationHook{%
  \ShellEscape{pdftops -eps tile-template.pdf
    tile-template.eps}%
  \ShellEscape{copy tile-template.eps \tgTileBaseIndx.eps}%
  \iftgfolder
    \ShellEscape{move \tgTileBaseIndx.eps \tgInFolder}%
  \fi
}
\afterPkgCreationHook{%
  \typeout{!! Package creation: \tgBaseName_package.pdf !!}}
```

Commentary. At insertion point of the hooks in the code stream, the tile file is named `tile-template`. The declared `\afterTileCreationHook` converts each tile file, to an EPS file of the same name. Then, it copies `tile-temp.eps` to `\tgTileBaseIndx.eps`. Next, we test whether this file came from a subfolder (using the switch `\iftgfolder`). Finally, if the file belongs in the folder, `\tgInFolder`, we move it there. For the `\afterPkgCreationHook`, we do nothing other than to write some text to the terminal, we use the base name of the graphic `\tgBaseName`.

`\tgTileBaseIndx`
`\iftgfolder`
`\tgInFolder`
`\tgBaseName`

`\packagesuffix{⟨name⟩}` The name used as suffix to the packaged tiled files. Originally, this was packaged, but I am changing it to package, to be in conformance with the acrosort package.

```
\newcommand{\packagesuffix}{package}
```

2.5 `\setTileParams`: A preamble command

`\setTileParams[⟨ig-opts⟩]{⟨nRows⟩}{⟨nCols⟩}{⟨path⟩}` The parameters are described in Section 1 (**Documentation**).

```
\newcommand{\setTileParams}[4][]{%
```

We require the entries in `#2` and `#3` (⟨*nRows*⟩ and ⟨*nCols*⟩) to be nonnegative natural numbers (1, 2, 3, . . .), so we pass through a dimension register and into a count register, this should make what is entered a natural number. If not greater or equal to 1, we complain, and set to the number 2.

```
\tg@dima #3\p@\relax
\ifdim\tg@dima < \p@
  \PackageWarning{tile-graphic}
    {Number of columns must be positive,\MessageBreak
     setting number of columns to 2}\tg@dima\tw@\p@\fi
\edef\nCols{\strip@pt\tg@dima}%
\tg@Cnt\nCols\relax
\edef\n@Cols{\the\tg@Cnt}\edef\nCols{\the\tg@Cnt}%
```

```
\tg@dima #2\p@\relax
\ifdim\tg@dima < \p@
  \PackageWarning{tile-graphic}
    {Number of rows must be positive,\MessageBreak
     setting number of rows to 2}\tg@dima\tw@\p@\fi
\edef\nRows{\strip@pt\tg@dima}%
\tg@Cnt\nRows\relax
\edef\n@Rows{\the\tg@Cnt}\edef\nRows{\the\tg@Cnt}%
\multiply\tg@Cnt \nCols\relax
\edef\nFilesCreated{\the\tg@Cnt}%
\def\pathToPic{#4}%
```

Parse the path to obtain the parts of the path, area, base, and extension.

```
\filename@parse{#4}%
\edef\tg@dir{\filename@area}%
\ifx\tg@dir\@empty\tgfolderfalse\else
  \expandafter\tg@rmslash\tg@dir
  \tgfoldertrue
\fi
\edef\tg@base{\filename@base}%
\edef\tgBaseName{\filename@base}%
\edef\tg@ext{\filename@ext}%
```

Get graphic dimensions, dimensions needed for TEX (pt) and PDF (bp)

```
\setbox\tg@box\hbox{\includegraphics[draft,#1]{#4}}%
\setlength\tg@dima{\the\wd\tg@box}%
\tg@dima=.99626\tg@dima
\divide\tg@dima \nCols
\edef\bpWdtile{\strip@pt\tg@dima}%
\setlength\tg@dima{\the\ht\tg@box}%
\tg@dima=.99626\tg@dima
\divide\tg@dima \nRows
\edef\bpHttile{\strip@pt\tg@dima}%
\setbox\tg@box\box\voidb@x
\tg@dima=\bpHttile pt \relax
\edef\tg@HT{\the\tg@dima}
\tg@dima=\bpWdtile pt \relax
\edef\tg@WD{\the\tg@dima}
```

Set margins and screen size using the web package. If `\ifpassthruTG` is true, we use reasonably size dimensions to display the **Tile Graphic Report**; otherwise, we use dimensions based on the size of the graphic determined by ⟨*path*⟩.

```
\ifpassthruTG
  \web@MargScrDimOpts{.25in}{.25in}{24pt}{.25in}{5in}{6in}
\else
  \web@MargScrDimOpts{0pt}{0pt}{0pt}{0pt}{\tg@HT}{\tg@WD}
\fi
```

Bounding box calculations for the tiles

The *y*-coordinate calculations: If `\nRows` is 3, for example, we calculate 4 *y*-

coodinates, \y1, \y2, \y3, \y4, from bottom to top.

```
  \@tempcnta\n@Rows\relax
  \advance\@tempcnta\@ne
  \edef\n@Rows{\the\@tempcnta}
  \@tempcnta\z@
  \tg@dima0pt
  \tg@dimb=\tg@HT\relax
  \@whilenum \@tempcnta < \n@Rows \do {%
    \advance\@tempcnta\@ne
    \csarg\edef{y\the\@tempcnta}{\strip@pt\tg@dima}
    \advance\tg@dima \tg@dimb
  }
```

The x-coordinate calculations If `\nCols` is 2, for example, we calculate 3 x-coodinates, \x1, \x2, \x3, from left to right.

```
  \@tempcnta\n@Cols\relax
  \advance\@tempcnta\@ne
  \edef\n@Cols{\the\@tempcnta}
  \@tempcnta\z@
  \tg@dima0pt
  \tg@dimb=\tg@WD\relax
  \@whilenum \@tempcnta < \n@Cols \do {%
    \advance\@tempcnta\@ne
    \csarg\edef{x\the\@tempcnta}{\strip@pt\tg@dima}
    \advance\tg@dima \tg@dimb
  }
```

Having finished the calculations, we then execute `\tg@wrtthefiledoc`.

```
  \if@wrtorpkg\expandafter
    \tg@wrtthefiledoc\fi
}
```

`\tg@wrtthefiledoc` This command writes the `tile-template.tex` file to the source file folder. It will read,

```
\RequirePackage{tmp}
\documentclass{article}
\usepackage{web}
\usepackage{graphicx}
\let\WriteBookmarks\relax
\margins{0pt}{0pt}{0pt}{0pt}
\screensize{\tg@HT}{\tg@WD}
\parindent0pt\parskip0pt
\begin{document}
\tgInputContent
\end{document}
```

This is the file that is compiled using `\ShellEscape` to create the individual tile files. This document contains a trick, the use of the `tmp` package, which is written dynamically just before this file is compiled. The `tmp` package is created by `\tg@wrttmppkg`, defined next. The `tmp` package defines the command `\tgInputContent` in the body of the document.

`\tgInputContent`

```
\def\tg@wrtthefiledoc{\newwrite\wrttiledoc
  \long\def\IWTD##1{\immediate\write\wrttiledoc{##1}}
  \immediate\openout \wrttiledoc tile-template.tex
  \IWTD{%
    \string\RequirePackage{tmp}^^J%
    \string\documentclass{article}^^J%
    \string\usepackage{web}^^J%
    \string\usepackage{graphicx}^^J%
    \string\let\string\WriteBookmarks\string\relax^^J%
    \string\margins{0pt}{0pt}{0pt}{0pt}^^J%
    \string\screensize{\tg@HT}{\tg@WD}^^J%
    \string\parindent0pt\string\parskip0pt^^J%
    \string\begin{document}^^J%
    \string\tgInputContent^^J%
    \string\end{document}}%
  \immediate\closeout \wrttiledoc
}
```

`\tg@wrttmppkg{`⟨*basename*⟩`}{`⟨*indx*⟩`}` Write the `tmp` package dynamically: ⟨*basename*⟩ is the base name of the graphic; ⟨*indx*⟩ is the index of the tile (`01`, `02`, `03`, ...). The action of this package is to define `\tgInputContent` to input the file

⟨*basename*⟩_⟨*indx*⟩`.cut`

```
\def\tg@wrttmppkg#1#2{\def\CommentCutFile{tmp.sty}%
  \immediate\openout\CommentStream \CommentCutFile
    \immediate\write\CommentStream{\string
      \def\string\tgInputContent{\string
      \InputIfFileExists{#1_#2.cut}%
      {}{\string\null}}}%
  \immediate\closeout\CommentStream
}
```

2.6 `\tileTheGraphic`: A document body command

The file `create-tb.tex` contains the single command `\tileTheGraphic` in the body of the document. It has not arguments.

```
\InputIfFileExists{package-data.cut}{}{}%
```

`\tg@msgi` is the content of `create-tg` to deliver the **Tile Graphic Report**. It may be redefined. This content command is expanded in `\tileTheGraphic`, defined below.

```
\def\tg@msgi{%
    \textbf{Tile Graphic Report}\medskip\par
    \textbf{Options:}
    \begin{itemize}
    \if@wrttofiles
      \item The \texttt{wrttofiles} option  taken,
    {\nFilesCreated} files written ({\nRows}~rows, {\nCols}~cols):
    \begin{quote}
    \texttt{\tg@base\_01.pdf}, \texttt{\tg@base\_02.pdf}, \dots.
    \end{quote}
    \ifx\tg@dir\@empty Files saved to source file folder. \else
    Files saved to the \texttt{\tgInFolder} folder.\fi
  \else
   \item The \texttt{!wrttofiles} option taken, no files written.
  \fi
  \if@packagefiles
  \item The \texttt{packagefiles} option taken,
   package file saved as \texttt{\tg@base\_\packagesuffix.pdf}.
   The package contains {\nFilesCreated} pages of tiled graphics.
   \ifx\tg@dir\@empty
    The package file saved to source file folder.
   \else
    The package file saved to the \texttt{\tgInFolder} folder.
   \fi
  \else
  \item The \texttt{!packagefiles} option taken,
   no package file created.
  \fi
  \end{itemize}
  \if@wrtorpkg\else
   Try using the \texttt{wrttofiles} and/or
   \texttt{packagefiles} option(s) to obtain a nontrivial result.
  \fi
}
```

`\tileTheGraphic` (No arguments) This is the command that tiles the graphic.

```
\def\tileTheGraphic{\begingroup\let\@nu\@nameuse
```

If the `!packagefiles` option is in effect, we set `\ifpassthruTG` to false so we can execute the `\else` part within `create-tg.tex` (as opposed to in ⟨*basename*⟩`_package.tex`).

```
 \if@packagefiles\else\global\passthruTGfalse\expandafter
  \tg@msgi % provide content
 \fi
 \ifpassthruTG
  \tg@msgi % provide content
 \else
```

Okay, we are here either because we are compiling this file either from within ⟨*basename*⟩`_package.tex` or from within the source file `create-tg.tex` with the `!packagefiles` option is in effect.

```
  \global\let\tg@IndxToks\@empty
```

Nested `\multido` loop to create grid

```
  \if@wrtorpkg\expandafter\tg@domultido\fi
 \fi
 \endgroup
```

```
  \xdef\nFilesCreated{\the\tg@Cnt}%
  \edef\x{\if@wrtorpkg\noexpand\compileTileFiles\fi}\x
}
```

`\tg@domultido` Called by `\tileTheGraphic`. Consists of nested `\multido` loops. The command both creates the package file and the individual tile graphics, depending on the options. The command `\tg@IndexToks` creates a token list of indices `{01}{02}{03}`... that is later used in a `\@tfor` loop.

```
\def\tg@domultido{%
    \tg@Cnt\z@
    \multido{\iR=\nRows+-1}{\nRows}{%
      \multido{\iC=1+1}{\nCols}{%
        \global\advance\tg@Cnt\@ne
        \ifnum\tg@Cnt<10\relax
          \edef\x{0\the\tg@Cnt}\else
          \edef\x{\the\tg@Cnt}\fi
          \edef\y{\noexpand\g@addto@macro\noexpand
            \tg@IndxToks{{\x}}}\y
          \@tempcntb\iC
          \advance\@tempcntb\@ne
          \edef\oX{\the\@tempcntb}%
          \@tempcntb\iR
          \advance\@tempcntb\@ne
          \edef\oY{\the\@tempcntb}%
```

We write the CUT files.

```
          \wrtTileCuts
```

Include the graphic with the appropriate viewport and clip; however, we executed the temporary command `\z` if the option `\packagefiles` is in effect.

```
          \edef\z{\noexpand\parbox{\tg@WD}{\noexpand
            \includegraphics[width=\tg@WD,%
              viewport=\@nu{x\iC} \@nu{y\iR} \@nu{x\oX} \@nu{y\oY},%
              clip]{\pathToPic}}}\if@packagefiles\expandafter
              \z\expandafter\newpage\fi
      }% inner multido
    }% outer multido
}
```

`\wrtTileCuts` (Called by `\tg@domultido`) The CUT files created are the body content of the `tile-template.tex` file. The CUTs are also used by the package file routine. The content of these CUT files has the following form:

```
\parbox{132.23935pt}{\includegraphics[width=132.23935pt,
   viewport=0 114.23943 132.23935 228.47885,clip]{⟨pathToPic⟩}}
```

where the values of `width` and `viewport` were calculated by the `\setTileParams` in the preamble.

```
\def\wrtTileCuts{%
  \def\CommentCutFile{\tg@base_\x.cut}%
  \immediate\openout\CommentStream=\CommentCutFile
  \immediate\write\CommentStream{\string
    \parbox{\tg@WD}{\string
    \includegraphics[width=\tg@WD,%
    viewport=\@nu{x\iC} \@nu{y\iR} \@nu{x\oX} \@nu{y\oY},%
    clip]{\pathToPic}}}\immediate\closeout\CommentStream
}
```

`\compileTileFiles` (Called from `\tileTheGraphic`) This command performs the `\ShellEscape` steps. It is executed only if the `wrttofiles` or `packagefiles` option is taken (or both).

```
\def\compileTileFiles{%
  \ifpassthruTG
    \if@packagefiles
```

Package the tile files. This code is executed by `create-tg.tex` (because `\passthruTG` is true). We write the `package-data.cut` file, later input by the `tmp` package, which puts `\passthruTG` to false.

```
      \def\CommentCutFile{package-data.cut}%
      \immediate\openout\CommentStream \CommentCutFile
        \immediate\write\CommentStream{\string\passthruTGfalse}%
      \immediate\closeout\CommentStream
```

Then copy `\create-tg.tex` to ⟨*basename*⟩`_package.tex`. Keep in mind that when we compile `\tg@base_package.tex` which uses the tile-graphic package, `\passthruTG` is false. When ⟨*basename*⟩`_package.tex` gets here, this block of code is skipped over.

```
      \ShellEscape{\tg@syscopy \jobname.tex
        \tg@base_\packagesuffix.tex}%
```

and compile with the `--shell-escape` switch,

```
      \iftg@direct
        \ShellEscape{\pkg@ppArgs{\tg@pdfcreator@app} --shell-escape
          \tg@base_\packagesuffix.tex}%
      \else
        \ShellEscape{\latex@ppArgs{latex} --shell-escape
          \tg@base_\packagesuffix.tex}%
        \ShellEscape{\dvips@ppArgs{dvips} \tg@base_\packagesuffix.dvi}%
        \ifnum\tg@case=\thr@@
          \ShellEscape{\pkg@ppArgs{\tg@pdfcreator@app}
          \tg@base_\packagesuffix.ps}%
        \else
          \ShellEscape{\pkg@ppArgs{\tg@pdfcreator@app} /N /Q
          "\tg@fullPathToSource/\tg@base_\packagesuffix.ps"}%
        \fi
        \ShellEscape{\tg@sysdel \tg@base_\packagesuffix.dvi
          \tg@base_\packagesuffix.ps}%
      \fi
```

`\@fterPkgCreationHook` Insert the after-package-creation-hook `\@fterPkgCreationHook`

```
      \@fterPkgCreationHook
```

clean up,

```
    \ShellEscape{\tg@sysdel \tg@base_\packagesuffix.tex
      \tg@base_\packagesuffix.log \tg@base_\packagesuffix.aux}%
```

and move into the folder from which the graphic resides, if necessary.

```
    \iftgfolder
      \ShellEscape{\tg@sysmove
        \tg@base_\packagesuffix.pdf \tgInFolder}\fi
   \fi
\else
```

Create the tile files. This block is compiled if `\passThruTG` is false and the `wrttofiles` option is taken. The block gets compiled in two instances:

1. by ⟨*basename*⟩_`package.tex` if the `packagefiles` and `wrttofiles` options are taken;
2. by `create-tg.tex` if `!packagefiles` and `wrttofiles` options are taken. (Recall that if `!packagefiles` is taken, then `\ifpassThruTG` is set to false earlier in the code stream.)

```
  \if@wrttofiles
    \edef\@tforexp{\noexpand
      \@tfor\noexpand\Indx:=\tg@IndxToks}%
```

For each token in `\tg@IndexToks` (`{01}{02}{03}...`), we compile the dynamic file `tile-template.tex`.

```
    \@tforexp \do {%
      \edef\tgTileBaseIndx{\tg@base_\Indx}%
```

Create the `tmp` package with parameters ⟨*basename*⟩ and ⟨*indx*⟩.

```
      \edef\x{\noexpand\tg@wrttmppkg{\tg@base}{\Indx}}\x
```

Compile this turkey,

```
      \iftg@direct
        \ShellEscape{\tile@ppArgs{\tg@pdfcreator@app}
          tile-template.tex}%
      \else
        \ShellEscape{\latex@ppArgs{latex} --shell-escape
          tile-template.tex}%
        \ShellEscape{\dvips@ppArgs{dvips} tile-template.dvi}%
        \ifnum\tg@case=\thr@@
          \ShellEscape{\pkg@ppArgs{\tg@pdfcreator@app}
            tile-template.ps}%
        \else
          \ShellEscape{\pkg@ppArgs{\tg@pdfcreator@app} /N /Q
          "\tg@fullPathToSource/tile-template.ps"}%
        \fi
        \ShellEscape{\tg@sysdel tile-template.dvi
          tile-template.ps}%
      \fi
```

`\@fterTileCreationHook` Insert the after-tile-creation-hook `\@fterTileCreationHook`

```
        \@fterTileCreationHook
        \ShellEscape{\tg@syscopy tile-template.pdf
          \tgTileBaseIndx.pdf}%
```

clean up,

```
        \ShellEscape{\tg@syscopy tile-template.pdf
          \tgTileBaseIndx.pdf}%
      }% do
```

and move to another folder if necessary

```
      \iftgfolder
        \ShellEscape{\tg@sysmove \tg@base_*.pdf \tgInFolder}\fi
    \fi
  \fi
```

finished! Just clean up all aux files.

```
  \ShellEscape{\tg@sysdel \tg@base_*.cut package-data.cut}%
  \ShellEscape{\tg@sysdel tmp.sty tile-template.*}%
}
```

Letting `\WriteBookmarks` to `\relax` prevents hyperref from complaining about the OUT file is not up to date. No bookmarks are created. Also load the configuration

`tg.cfg` file `tg.cfg`, if it exists.

```
\let\WriteBookmarks\relax
\InputIfFileExists{tg.cfg}{}{}
\tg@restoreCats
\parindent0pt
⟨/package⟩
```

3 Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

4 Change History

v1.0 (2020/05/27)

General: Set version number to v1.0 to publish package 6

v1.0.1 (2020/05/27)

General: Replace suffix of packaged with just package, but control the naming with `\packagesuffix`. 10