The external package*

Michael D. Adams
https://michaeldadams.org

Abstract

The `external` package allows you to include the result of rendering LaTeX code in a separate document that has its own preamble. This is useful when you want to use symbols from a package that you do not want to load into your main document. For example, your main document may use symbols from multiple packages that conflict or otherwise cannot be loaded together.

Contents

*This document corresponds to external v0.1, dated 2019/01/05.

1 Introduction

The `external` package allows you to include the result of rendering LaTeX code in a separate document that has its own preamble. For example, you may want to use symbols from packages that cannot be loaded at the same time. By rendering them in separate documents and then including their rendered results in a master document, this allows you to use both symbols without conflicts.

This document serves as both the documentation and test suite for the `external` package.

1.1 Quick Start

Suppose you want to use the `textbeta` symbol from the `textgreek` package, but you do do not want to load the `textgreek` package into your master document. (Maybe it is incompatible with some other package that you use or maybe LaTeX has run out of space for declaring new fonts.) You can render `textbeta` using the `external` command as in the following.

```
An atom undergoing
\external[preamble={\usepackage{textgreek}}]{\textbeta-decay}
can emit an electron.
```

An atom undergoing β-decay can emit an electron.

If you prefer, you can also use the environment `externalenv` → P. 5 as in the following. (The only difference between the `external` command and the `externalenv` → P. 5 environment is that one is a command and the other is an environment.)

```
An atom undergoing
\begin{externalenv}[preamble={\usepackage{textgreek}}]
\textbeta-decay
\end{externalenv}
~can emit an electron.
```

An atom undergoing β-decay can emit an electron.

If you want to format the LaTeX code as "display" math, use the `math=display` option as in the following example.

```
The solution to the two dimensional integral
\external[preamble={\usepackage{amsmath}}, math=display]
  {\iint xy\,dx\,dy}
is involves $x^2$ and $y^2$.
```

The solution to the two dimensional integral

$$\iint xy\,dx\,dy$$

is involves x^2 and y^2.

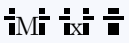
Note that even though these examples are rendered as separate LaTeX documents, they are automatically properly spaced relative to the surrounding text. In the resulting PDF, they even behave properly with regard to copy-and-paste.

See Section 3 more details about options and Section 6 for common issues and their workarounds.

1.2 Notation

For debugging purposes, this documentation surrounds most examples with `<` and `>` which render as and . These are used as a gauge or registration mark. This makes it easy to see whether there is extra space to the left or right of a symbol and whether parts of the symbol extend below the baseline. The bottom of the bottom bar is at the baseline. The top of the top, middle and bottom bars are at the font size (10 points), where the top of an "M" would occur, and where the top of an "x" would occur, respectively. This is demonstrated in the following example.

```
<M> <x> <>
```



1.3 Related Packages

TODO

Main difference is ability to specify own preamble and exact sizing

`childdoc` TODO: describe

`combine` TODO: describe

`docmute` TODO: describe

`includex` TODO: describe

`minidocument` TODO: describe

`newclude` TODO: describe

`pgfplots` TODO: describe

`preview` TODO: describe

`standalone` TODO: describe

`subdocs` TODO: describe

`subfiles` TODO: describe

`tcolorbox` TODO: describe

`tikz` TODO: describe

`example` TODO: describe

`examplep` TODO: describe

`latexdemo` TODO: describe

`showexpl` TODO: describe

2 High-level Commands

2.1 `external` and `externalenv`

The main commands of this package are `external` and `externalenv`.

`\external[`⟨*options*⟩`]{`⟨*code*⟩`}`

Externally renders the the LaTeX code in ⟨*code*⟩. An examples of its usage is the following.

```
<\external[preamble={\usepackage{amsmath}}, math=inline]
   {\iint xy\,dx\,dy}>
```

$$\iint xy\,dx\,dy$$

\begin{externalenv}[⟨*options*⟩]
  ⟨*environment content*⟩
\end{externalenv}

The same as `external`. The only difference is that `external` is a command and `externalenv` is an environment. An example of its usage is the following.

```
<\begin{externalenv}[preamble={\usepackage{amsmath}}, math=inline]
   \iint xy\,dx\,dy
 \end{externalenv}>
```

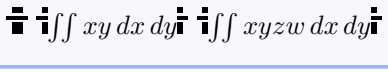
2.2 `newexternal` and `newexternalenv`

It is sometimes useful to define versions of `\external` [→ P. 4] and `externalenv` that have default values for their options. These can be created with `newexternal` and `newexternalenv`. For example, you might want to define versions that load the `amsmath` package by default.

\newexternal[⟨*options*⟩]{⟨*command name*⟩}

With `newexternal`, one can define a version of `\external` [→ P. 4] that has default values for its options.

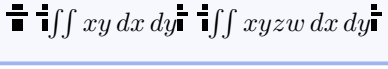
```
<\newexternal[preamble={\usepackage{amsmath}}, math=inline]{\ams}>
<\ams{\iint xy\,dx\,dy}>
<\ams[math=false]{$\iint xyzw\,dx\,dy$}>
```



\newexternalenv[⟨*options*⟩]{⟨*command name*⟩}

With `newexternalenv`, one can define a version of `externalenv` that has default values for its options.

```
<\newexternalenv[preamble={\usepackage{amsmath}}, math=inline]{amsenv}>
<\begin{amsenv}\iint xy\,dx\,dy\end{amsenv}>
<\begin{amsenv}[math=false]$\iint xyzw\,dx\,dy$\end{amsenv}>
```



3 Options

Options that are passed to commands are parsed using the `keyval` package. Their syntax is of the form:

$$[\langle key_1\rangle=\langle value_1\rangle,\ \langle key_2\rangle=\langle value_2\rangle,\ \cdots,\ \langle key_n\rangle=\langle value_n\rangle]$$

\externalkeys{⟨*options*⟩}

You can set the default value for any options with the `externalkeys` command. For example, if want to default to use `mypdflatex` instead of `pdflatex` to compile LaTeX code, you could add the following command.

```
\externalkeys{latex=mylatex}
```

You can also specify this by passing the option when the `external` package is loaded as seen in the following example.

```
\usepackage[latex=mylatex]{external}
```

3.1 High-level Options

documentclass=⟨*class*⟩ (initially `article`)

This option specifies (by way of `documentclass`) the document class to be used by the intermediate LaTeX file that is generated for each piece of externally rendered code. For example, the following uses the `proc` class, which (unlike `article`) contains the `pagename` macro.

```
<\external[documentclass=proc]{\pagename}>
```

If the value of this key is blank, a `documentclass` declaration is not added to the intermediate file. In this case, you will likely want to put a `documentclass` declaration in the `preamble` → P. 7 option as in the following example. (Though this could be achieved with `documentclass=prog`, so doing it this way is gratuitous and solely for the sake of an example.)

```
<\external[documentclass={}, preamble={\documentclass{proc}}]
   {\pagename}>
```

documentclass/options=⟨*options*⟩ (initially empty)

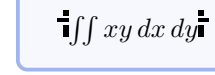
This options specifies any options to pass to the document class to be used by the LaTeX file that is generated for externally rendered code. For example, the following specifies passing the `12pt` option to `article`, which changes the default font to be 12 points tall.

```
<\external[documentclass/options={12pt}]{M}>
```

`preamble`=⟨*code*⟩ (initially empty)

This options specifies LaTeX code to be put in the preamble of the intermediate LaTeX file that is generated for externally rendered code. For example, you might want to load packages as in the following.

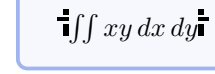
```
<\external[preamble={\usepackage{amsmath}}]{$\iint xy\,dx\,dy$}>
```



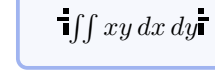
`math`=⟨*`false`, `inline`, or `display`*⟩ (initially `false`)

This option determines whether the body of `\external` → P. 4 or `externalenv` → P. 5 is treated as math, and if so, whether it is inline math or display math. The following demonstrate each value possible for this option.

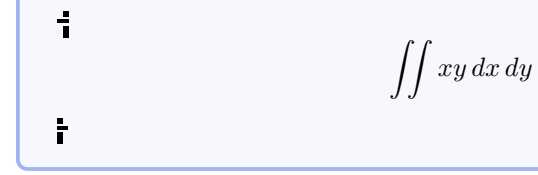
```
<\external[preamble={\usepackage{amsmath}}, math=false]
   {$\iint xy\,dx\,dy$}>
```



```
<\external[preamble={\usepackage{amsmath}}, math=inline]
   {\iint xy\,dx\,dy}>
```



```
<\external[preamble={\usepackage{amsmath}}, math=display]
   {\iint xy\,dx\,dy}>
```



Note that `math=display`, as seen in the following example, is equivalent to the incantation `\[\external{$\displaystyle...$}\]`.

```
<\[\external[preamble={\usepackage{amsmath}}]
     {$\displaystyle\iint xy\,dx\,dy$}\]>
```

`margin/top`=⟨*length*⟩ (initially `1in`)

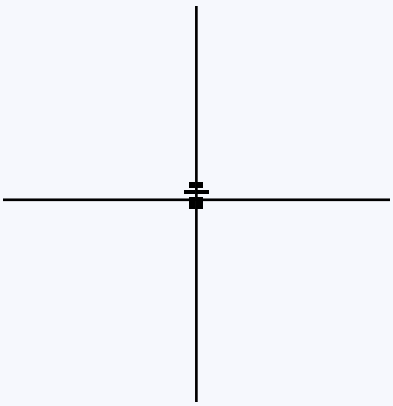
`margin/bottom`=⟨*length*⟩ (initially `1in`)

`margin/left`=⟨*length*⟩ (initially `1in`)

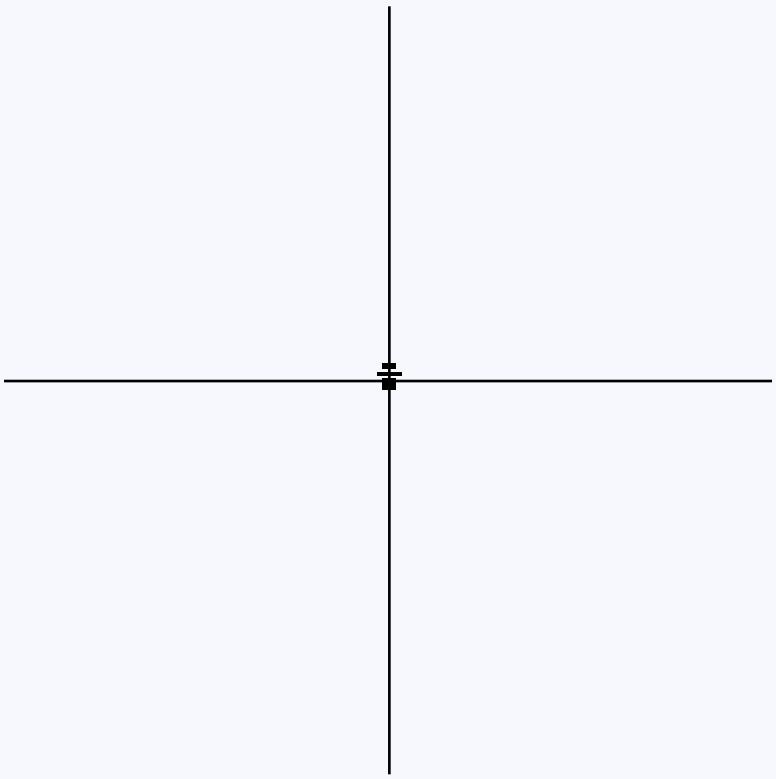
`margin/right`=⟨*length*⟩ (initially `1in`)

These options specify the margin to place around the LaTeX code being rendered externally. This is useful if the LaTeX code being rendered externally draws outside its bounding box. If there is not enough margin to contain the drawn portions, the result may be clipped. For example, compare the two following examples. The 2 inch rules are clipped when the default margins are used but are not clipped when 3 inch margins are used.

```
\vspace{2in}
\hspace{2in}
<\begin{externalenv}
   \rlap{\rule[3pt]{2in}{1pt}}%
   \llap{\rule[3pt]{2in}{1pt}}%
   \smash{\rule[-2in]{1pt}{4in}}%
 \end{externalenv}>
\vspace{2in}
```



```
\vspace{2in}
\hspace{2in}
<\begin{externalenv}[margin/top=3in, margin/bottom=3in,
                     margin/left=3in, margin/right=3in]
   \rlap{\rule[3pt]{2in}{1pt}}%
   \llap{\rule[3pt]{2in}{1pt}}%
   \smash{\rule[-2in]{1pt}{4in}}%
 \end{externalenv}>
\vspace{2.2in}
```



4 Low-level Options

`before`=⟨*code*⟩ (initially `\stepcounter{external@number}`)

This option specifies LaTeX code to run before the rest of the code in an `\external`→ P. 4 command or `externalenv`→ P. 5 environment. This is particularly is useful for incrementing any counters used in the `file`→ P. 13 option. The following is an example of this in action.

```
\newcounter{foo}%
\thefoo
<\external[file=external-external-before,
           before={\stepcounter{foo}}]{}>%
\thefoo
```


`before compile`=⟨*code*⟩ (initially empty)

This option specifies LaTeX code to run before compiling the externally rendered code in an `\external` → P. 4 command or `externalenv` → P. 5 environment. The following is an example of this in action.

```
\newcounter{baz}%
\thebaz
<\external[file=external-external-before,
           before compile={\stepcounter{baz}}]{}>%
\thebaz
```


`before read`=⟨*code*⟩ (initially empty)

This option specifies LaTeX code to run before reading the file output by the externally rendered code in an `\external` → P. 4 command or The following is an example of this in action.

```
\newcounter{quux}%
\thequux
<\external[file=external-external-before,
           before read={\stepcounter{quux}}]{}>%
\thequux
```


`after`=⟨*code*⟩ (initially empty)

This option specifies LaTeX code to run after the rest of the code in an `\external` → P. 4 command or `externalenv` → P. 5 environment. The following is an example of this in action.

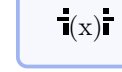
```
\newcounter{bar}%
\thebar
<\external[file=external-external-after,
           after={\stepcounter{bar}}]{}>%
\thebar
```

`before savebox`=⟨*code*⟩ (initially empty)

This option specifies code to be put before the `savebox` that is used in the intermediate LaTeX file that is generated for externally rendered code. This option corresponds to the ⟨*before savebox*⟩ argument of `\ExternalCode` → P. 15. This option is rarely needed.

The following example demonstrates the use of this option, though since the definition of `p` could be put in the preamble, putting it in `before savebox` is gratuitous and solely for the sake of an example.

```
<\external[before savebox={\newcommand{\p}[1]{(##1)}}]{\p{x}}>
```

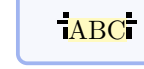


`before usebox`=⟨*code*⟩ (initially empty)

This option specifies code to be put before the `usebox` that is used in the intermediate LaTeX file that is generated for externally rendered code. This option corresponds to the ⟨*before usebox*⟩ argument of `\ExternalCode` → P. 15. This option is rarely needed.

The following example demonstrates the use of this option, though since there are other ways to accomplish this effect, using `before usebox` is gratuitous and solely for the sake of an example. Note that we have to set the margins to small or zero lengths to prevent them from overlapping the rest of the page.

```
<\external[preamble={\usepackage{xcolor}},
           before usebox={\pagecolor{yellow!30}},
           margin/top=1pt, margin/bottom=1pt,
           margin/left=0pt, margin/right=0pt]
  {ABC}>
```



`latex`=⟨*program name*⟩ (initially empty)

This option specifies the program to use to compile the intermediate LaTeX file that is generated for each bit of externally rendered code.

Blank means to autodetect between `pdflatex`, `xelatex`, or `lualatex` to match whatever the master document is being compiled with.

For example, if you wanted to force certain code to run under `pdflatex`, you could do the following.

```
<\external[latex=pdflatex]{\pdfescapehex{ABC}}>
```

`latex/options`=⟨*code*⟩ (initially `-halt-on-error -interaction=batchmode`)

This option specifies what command line options to pass to LaTeX when compiling the intermediate LaTeX file that is generated for externally rendered code.

Note that if you change this, you will almost certainly want to include the `-halt-on-error` and `-interaction=batchmode` options in whatever you change it to.

For example, the `ifplatform` package needs the `-shell-escape` option in order to give precise platform information. This can be specified as in the following.

```
<\external[preamble={\usepackage{ifplatform}}]
   {\platformname}>
<\external[preamble={\usepackage{ifplatform}},
           latex/options={-shell-escape -halt-on-error
                          -interaction=batchmode}]
   {\platformname}>
```

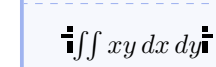
Note that if this document was compiled on Windows, then the two calls to `\external` → P. 4 in this example will produce the same results as each other, but on any other platform they will be different.

`file`=⟨*file basename*⟩(initially `\jobname-external-\arabic{external@number}`)

This option specifies the basename of the intermediate files that are generated for externally rendered code.

For example, the following uses `external-external-file` as the basename.

```
<\external[file=external-external-file,
           preamble={\usepackage{amsmath}}]
   {$\iint xy\,dx\,dy$}>
```



Be careful not to use the same filename for two different pieces of externally rendered code as that can lead to unexpected results.

`file/tex`=⟨*extension*⟩ (initially `.tex`)

`file/dim`=⟨*extension*⟩ (initially `.dim`)

`file/pdf`=⟨*extension*⟩ (initially `.pdf`)

These options specify the extensions to use for the intermediate LaTeX, dimension, and PDF files, respectively.
These options are rarely needed.
An example of using this is the following.

```
\DeclareGraphicsRule{.mypdf}{pdf}{.mypdf}{}%
<\external[
   file/tex=.mytex, file/dim=.mydim, file/pdf=.mypdf,
   file=external-external-extension,
   before read={\ShellEscape{mv external-external-extension.pdf
                                external-external-extension.mypdf}}]
   {ABC}>
```

`includegraphics/options`=⟨*key-value sequence*⟩ (initially empty)

This option specifies options to be passed to the `includegraphics` command that is used to read into the master document the result of rendering the LaTeX code that is rendered separately. For example, the following uses `angle` to rotate the image read by `includegraphics`.

```
<\external[includegraphics/options={angle=45}]{M}>
```

`debug`=⟨*`true` or `false`*⟩ (initially `false`)

Whether to print tracing information to standard out. This is helpful in determining exactly what part of a command failed.
For example, consider the following call to `\external` → P. 4.

```
<\external[preamble={\usepackage{amsmath}}]{$\iint xy\,dx\,dy$}>
```

When the `debug` option is `true`, lines like the following will be printed to the standard output.

```
**** Begin \ExternalWrite on {external-external-21}
       with {$\iint xy\,dx\,dy$}
**** End \ExternalWrite on {external-external-21}
**** Begin \ExternalCompile on {external-external-21}
**** End \ExternalCompile on {external-external-21}
**** Begin \ExternalRead on {external-external-21}
**** End \ExternalRead on {external-external-21}
```

5 Low-level Commands

\ExternalWrite[⟨*options*⟩]{⟨*body*⟩}

\ExternalCompile[⟨*options*⟩]

\ExternalRead[⟨*options*⟩]

Both the `\external` → P. 4 command and the `externalenv` → P. 5 environment are broken up into three phases:

1. writing the intermediate LaTeX file,
2. compiling the intermediate LaTeX file into a PDF file, and
3. reading the resulting intermediate PDF file.

These are handled with `\ExternalWrite`, `\ExternalCompile`, and `\ExternalRead`, respectively. For example, instead of using `\external` → P. 4, you could explicitly call each of these as in the following.

```
<\ExternalWrite[file=external-external-separate,
                preamble={\usepackage{amsmath}}]
   {$\iint xy\,dx\,dy$}>
<\ExternalCompile[file=external-external-separate]>
<\ExternalRead[file=external-external-separate]>
```

Taking explicit control of these is particularly useful if you want to cache compilation results. See the `\ExternalCode` command for an example of this.

\ExternalCode{⟨*dimension file*⟩}{⟨*preamble*⟩}{⟨*before savebox*⟩}{⟨*body*⟩}{⟨*before usebox*⟩}

This command expands to the code used in the intermediate LaTeX file. It is useful if you want to store the LaTeX code to be rendered in a separate file and reuse the compiled results between compilations of the master LaTeX file. The ⟨*dimension file*⟩ is the full filename of the dimension file to be generated. The ⟨*preamble*⟩, ⟨*before savebox*⟩, and ⟨*before usebox*⟩ are the same as the corresponding options in Section 3. The ⟨*body*⟩ is the LaTeX code to be rendered.

For example, you might write the following standalone file.

external-standalone-simple.tex

```
\RequirePackage{external.code}
\ExternalCode
  {external-standalone-simple.dim}
  {\documentclass{article}\usepackage{amsmath}}
  {}
  {$\iint xy\,dx\,dy$}
  {}
```

Then in your master file you can compile and read that standalone file with the following commands.

```
<\ExternalCompile[file=external-standalone-simple]>
<\ExternalRead[file=external-standalone-simple]>
```

$$\iint xy\,dx\,dy$$

Be careful if you rename a standalone file, as you will need to change the ⟨*dimension file*⟩ argument to match. Otherwise, you will get an error along the lines of `In \ExternalRead, input dimension file does not exist`. Also note that `\ExternalCode` → P. 15 is defined in the `external.code` package. This package is imported by the main `external` package, so you do not necessarily need to import it separately. However, `external.code` is designed be minimal and has dependencies. Thus in the previous example, by doing `\RequirePackage{external.code}` instead of `\RequirePackage{external}` we minimize the compilation time. When there are a large number of standalone files, this difference can amount to a significant amount of time.
If you wanted to reuse compiled results between compilations of the master LaTeX file, you would want to manually run the following command.

```
pdflatex -shell-escape external-standalone-simple.tex
```

Then you would omit the call to `\ExternalCompile` → P. 15 and just call `\ExternalRead` → P. 15 as in the following.

```
<\ExternalRead[file=external-standalone-simple]>
```

$$\iint xy\,dx\,dy$$

6 Issues and Workarounds

6.1 Render not updating

A common issue is when changing the LaTeX code to be externally rendered (e.g., the ⟨*body*⟩ of a `\external`→P. 4), but those changes are not reflected in the master document after re-compiling the master document. The cause of this is that `external` has no way to detect whether compiling the intermediate LaTeX file succeeded or failed. A failure can happen for example if the LaTeX code to be externally rendered contains an error that causes the compilation of the intermediate LaTeX file to fail. If compilation of the intermediate LaTeX file fails, the dimension file and PDF file from previous compilation will not be overwritten.

To fix this, delete the PDF and dimension files. Then failure of the compilation of the intermediate LaTeX file will cause a file-not-found error when the PDF and dimension file are read. You can then fix the error in the LaTeX code to be externally rendered and use this file-not-found error to let you know when you have fixed that LaTeX code.

6.2 Hashes and command arguments

The use of hashes that one would use when referencing a command argument (e.g., `#1`) can lead to problems. For example, the following would lead to an error.

```
<\external[preamble={\newcommand{\p}[1]{(#1)}}]{\p{x}}>
```

The solution to this is to use double hashes (for example, `##1` instead of `#1`) as demonstrated in the following.

```
<\external[preamble={\newcommand{\p}[1]{(##1)}}]{\p{x}}>
```

```
<\begin{externalenv}[preamble={\newcommand{\p}[1]{(##1)}}]
   \p{x}
 \end{externalenv}>
```

This even applies in the body of an `\external`→P. 4 or `externalenv`→P. 5 as seen in the following.

```
<\external{\newcommand{\p}[1]{(##1)}\p{x}}>
```

```
<\begin{externalenv}
   \newcommand{\p}[1]{(##1)}%
   \p{x}
 \end{externalenv}>
```

This also applies to standalone files as seen in the following.

external-standalone-hash.tex

```
\RequirePackage{external.code}
\ExternalCode
  {external-standalone-hash.dim}
  {\documentclass{article}\newcommand{\p}[1]{(##1)}}
  {}
  {\p{x}}
  {}
```

```
<\ExternalCompile[file=external-standalone-hash]>
<\ExternalRead[file=external-standalone-hash]>
```

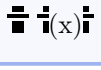
Finally, \newexternal →P. 5 and \newexternalenv →P. 5 require *four* hashes due to an extra level of indirection occurring in them as demonstrated in the following examples.

```
<\newexternal[preamble={\newcommand{\p}[1]{(####1)}}]{\paren}>
<\paren{\p{x}}>
```

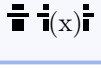
```
<\newexternalenv[preamble={\newcommand{\p}[1]{(####1)}}]{paren}>
<\begin{paren}\p{x}\end{paren}>
```

However, in their bodies this does not apply and only two hashes should be used as demonstrated in the following.

```
<\newexternal{\paren}>
<\paren{\newcommand{\parens}[1]{(##1)}\parens{x}}>
```



```
<\newexternalenv{paren}>
<\begin{paren}
   \newcommand{\parens}[1]{(##1)}%
   \parens{x}
 \end{paren}>
```



6.3 Category codes

Some commands change the category codes of many characters. These pose a problem for use with commands from this `external` package as the arguments to commands are parsed before those category codes have changed. The way to work around this is to use the `scantokens` macro to cause parts of those arguments to be re-parsed.

For example, the `DeclareFontShape` macro redefines category codes for characters used in its argument. Thus to use it one must insert a call to `scantokens` as in the following.

```
<\external[preamble={
             \usepackage{pifont}
             \DeclareFontFamily{U}{msa}{}
             \scantokens{
               \DeclareFontShape
                 {U}{msa}{m}{n}
                 {<-6>msam5<6-8>msam7<8->msam10}{}\relax}}]
   {\Pisymbol{msa}{15}}>
```

The `relax` before the end of the argument to `scantokens` ensures that `scantokens` does not insert an extra space at the end. See `https://tex.stackexchange.com/questions/117906/use-of-everyeof-and-endlinechar-with-scantokens` for details.

This trick also works when using a standalone file as in the following.

external-standalone-catcode.tex

```
\RequirePackage{external.code}
\ExternalCode
  {external-standalone-catcode.dim}
  {\documentclass{article}
   \usepackage{pifont}
   \DeclareFontFamily{U}{msa}{}
   \scantokens{
     \DeclareFontShape
       {U}{msa}{m}{n}
       {<-6>msam5<6-8>msam7<8->msam10}{}\relax}}
  {}
  {\Pisymbol{msa}{15}}
  {}
```

```
<\ExternalCompile[file=external-standalone-catcode]>
<\ExternalRead[file=external-standalone-catcode]>
```

Index