A guide to using the jdrlisting package in conjunction with the listings package*

Jim Ratliff†

Contents

---

* v. 1.0, 10/10/2018. This documentation is available on Overleaf, <tinyurl.com/jdrlstingDocs1>.

† jim@jim-ratliff.name, http://virtualperfection.com/jim.

List of Tables

List of Figures

Listings

I Introduction

The listings package, maintained by Jobst Hoffman, is:[1]

> a source code printer for LaTeX. You can typeset stand alone files as well as listings with an environment similar to verbatim as well as you can print code snippets using a command similar to \verb.

The jdrlisting package is Jim Ratliff's package to customize his usage of the the listings package.

In its simplest use, the document author need only ⓐ load the jdrlisting package[2] via \usepackage{jdrlisting} and then ⓑ issue the following command after \begin{document}:[3]

```
\initializeLaTeXjdrLst
```
⟨✂1⟩

in order to change the way that code is formatted and syntactically highlighted by the \lstinline command and lstlisting environment provided by the listings package.

The jdrlisting package provides two environments:[4]

- the jdrCodeSnip environment allows snippets or short segments of code[5] to be displayed like an equation and referred to by number like an equation, as in code snippet ✂2 shortly below.[6] This environment can envelop either ⓐ one or more \lstinline commands or ⓑ a lstlisting environment that displays a, preferably short, segment of code.
- the jdrLstListing environment facilitates using the lstlisting environment to typeset code that itself contains instances of the lstlisting environment.[7]

---

1 Carsten Heinz, Brooks Moses, and Jobst Hoffmann, "The Listings Package," September 2, 2018, version 1.7, tinyurl.com/listingsDocs, hereafter "ListingsDocs."

2 See section III.A.1.

3 See section III.B.

4 The jdrlstfloat environment is deprecated; see section XI.C.

5 See Appendix A.A for the distinction between a snippet of code and a segment of code.

6 See section III.E.

7 See section IV.C.2.2.

The jdrlisting package provides nine commands to facilitate document authors to format code identifiers that occur outside the verbatim contexts provided by the \lstinline command and the lstlisting environment.[8] Eight of these commands permit the document author to match the formatting/highlighting that listings/jdrlisting provides within \lstinline and lstlisting contexts to terms outside of \lstinline and lstlisting contexts.

In addition, this package provides the command \descriptorStylejdrLst to allow the document author to easily consistently style metacode, such as a descriptor of what type of argument is expected, for example, as in:[9]

```
\jdrHlineTC{length}{color}
```
⟨✂2⟩

I especially commend section IV for attention, even though it not specific to the jdrlisting package. It discusses several issues that come up when using the listings package and how to deal with them. In particular, section IV.A.1 discusses a leading cause of common, but hard to track down, fatal compilation errors: Not properly "escaping" particular characters ({}#%\) that occur in verbatim code that is being typeset by the \lstinline command located in a footnote, sectioning command, caption, or otherwise in the argument of another command.

Going further than what has been summarized directly above likely requires some understanding of my particular conventions of how I assign what type of identifiers to which particular classes of categories of identifiers. See section VII.

Up to this point,[10] I have used the listings package to typeset only LaTeX code. Thus, the functionality of my jdrlisting package and the focus of this discussion will be biased toward the typesetting of only LaTeX code.

Relatedly, I am not concerned with how to define a new language or dialect from scratch. However, I am certainly interested in how to amend (particularly, how to supplement) an existing language/dialect definition.[11]

---

8 See section V.

9 See section V.F.

10 At least as of October 10, 2018.

11 See sections VIII and IX.

The organization of this document is complementarily described in section II.

II Organization of this document

I have organized this document to be useful:

1. As a cookbook quick-reference for the document author who wants to quickly implement a code listing in a document; see in particular:
   - Section III: Overview of using the jdrlisting package. The many subsections of section III should provide you with a step-by-step guide to generating the listings you need, using the default formatting specified by the jdrlisting package.
   - Section IV: Some arcane features of listings you're likely to bump into (*ouch!*).
   - Section V: Commands to emulate or extend listings syntax highlighting in text outside of \lstinline or lstlisting-type environments.
2. As a guide for a document author wanting to customize listings; see in particular:
   - Section VI: Specially emphasizing the identifiers from a particular package when writing documentation about that package.
   - Section VII: My scheme of identifier-category classes and what identifiers to assign to each of them.
   - Section VIII: Commands to add or delete a list of identifiers to a class or to move those identifiers between classes.
   - Section IX: Managing identifiers.
   - Section X: Customizing the appearance of elements of listings whose appearance is controlled by jdrlisting.
3. As a resource for the advanced document author and the LaTeX programmer by providing at least hints if not a foundation for a deeper understanding to facilitate either advanced usage and/or changes in LaTeX code; in particular, see
   - Section XI: Discussion of selected implementation details of the jdrlisting package.

In some sense I treat an understanding of the standard listings package as if I consider it a prerequisite because I have delayed my review of how that package works until Appendix A: Selective summary of features and usage of the listings package.

III Overview of using the jdrlisting package

III.A Acquire and load jdrlisting; assess dependencies

III.A.1 Acquire and load jdrlisting

The first step is to load the jdrlisting package by including the following command in the preamble of your document:

```
\usepackage{jdrlisting}
```
⟨✂3⟩

As of the current date, the jdrlisting package is not included in any standard LaTeX package or even on The Comprehensive TeX Archive Network (CTAN). You will need to manually incorporate it into the files your document can access. It is available from, and issues can be reported at, github.com/jimratliff/jdrlisting.

III.A.2 Dependencies arising from loading the jdrlisting package

The jdrlisting package requires, and ensures the loading of, the following packages:

- the listings package, of course;
- the xcolor package, to support specifying colors for syntactical highlighting;
- the float package[12] to support the definition of the jdrlstfloat environment;[13]
- the jdrunicode package. This reliance is "soft," and could be eliminated by incorporating a small bit of code extracted from the jdrunicode package.[14]

In particular, note that the jdrlisting package loads the listings package with the final option,[15] viz., \RequirePackage[final]{listings}, ensuring that each list-

[12] The float package "[i]mproves the interface for defining floating objects such as figures and tables. Introduces the boxed float, the ruled float and the plaintop float. You can define your own floats and improve the behaviour of the old ones. The package also provides the H float modifier option of the obsolete here package." Its documentation is: Anselm Lingnau, "An Improved Environment for Floats," November 8, 2001.

[13] See section XI.C. Note that I have essentially deprecated the jdrlstfloat environment because I have not been able to discern any advantages of using it vis-à-vis using the float, caption, and label keys for lstlisting (or jdrLstListing) environment.

[14] The jdrunicode package is required only in order to support the following command: \newcommand{\jdrCodeSnipCharacter}{\jdrsymbol{✂}}.

[15] Although the documentation for listings asserts only (ListingsDocs at § 2.2 on page 11) that the draft option "prints no *stand alone* files, but shows the captions and defines the corresponding labels,'' [emphasis added] my empirical experience is that draft mode inhibits *all* lstlisting listings, even if the code is integrated in the document rather than standalone.

ing will be printed even if \documentclass is called with the global draft option, e.g., \documentclass[draft]{article}.[16,17]

The xcolor package is loaded without specifying any options. If you load any packages that require that xcolor be loaded with specific options, e.g., dvipsnames or x11names, you should ensure that those load before this package. If your document loads xcolor with any options, make sure you load xcolor before you load this package.

III.B Use the \initializeLaTeXjdrLst command to configure listings with the settings specified by jdrlisting

Once this package is loaded (see section III.A.1), the next step is to issue the following command in the body of the document, i.e., after \begin{document}:

```
\initializeLaTeXjdrLst
```
⟨✂4⟩

In its simplest use, this is all the document author need do in order to change the way that code is formatted and syntactically highlighted by the \lstinline command and lstlisting environment provided by the listings package.

More specifically, the \initializeLaTeXjdrLst command:[18]

- assigns colors for syntactical highlighting for distinct groups of identifiers (i.e., distinct groups of terms to be highlighted). See Table 1 for examples of the default formatting of different types of identifiers.
- expands the set of defined LaTeX-related identifiers for highlighting to include: ⓐ non-command identifiers in standard LaTeX, e.g., package and environment

[16] See Arash Esbati's answer to "How can I show listings even when the class option 'draft' is set?," TeX Stack Exchange, May 20, 2015.

[17] If you have trouble with inhibited listings because some other package is loading listings without final prior to your loading jdrlisting, you can either ⓐ change the loading order so that the non-final loading comes later or ⓑ insert \PassOptionsToPackage{final}{listings} before your \documentclass command (see, relatedly, Heiko Oberdiek's answer to "Applying options to already loaded package," TeX Stack Exchange, July 15, 2013).

[18] There are alternatives to \initializeLaTeXjdrLst that allow for fine control over which of my own packages are considered as ⓐ typical third-party packages, and therefore formatted identically to all other third-party packages, or instead ⓑ distinguished, allowing their identifiers to receive unique formatting. See [[section]].

names,[19] and ⓑ commands and non-command identifiers in third-party packages.[20]

◇ In particular, my own packages—such as this jdrlisting package and my jdrunicode, jdrhcline, and jdrsgame packages—are third-party packages, I define the command and non-command identifiers used in those packages.

- specifies formatting for code blocks produced by the lstlisting environment, e.g., sets margins, specifies that line numbers should be displayed and the format of those numbers, and specifies a background color to set off the code block from surrounding text.

III.C The default jdrlisting color scheme for different types of identifiers

The execution of the \initializeLaTeXjdrLst command implements jdrlisting's customizations of the colors with which different groups of identifiers will be formatted.[21] Table 1 shows an exemplar identifier for each type of identifier as an example of how each type of identifier is formatted by jdrlisting.[22]

III.D Typeset short snippets of code in-line within a paragraph using the \lstinline command

You can typeset a snippet of verbatim code in-line with a paragraph using the \lstinline command. Rather than using {...} as delimiters for its argument, you ① select a single character that does not appear within the string of verbatim code and then ② use that character as both the opening and closing of the argument.

In the below example, code snippet ✂5, ⓐ the verbatim string to be typeset is x&\#{}! and the delimiting character chosen is |:

```
\lstinline|x&\#{}!|
```
⟨✂5⟩

[19] [[Acknowledge that there was an apparently ineffective attempt to include these in the listings definitions but, at least for the LaTeX dialect, these do not wind up highlighted by default.]]

[20] See section VII.

[21] See section III.B.

[22] I omit keywordstyle identifiers from the table because, as I explain in section IX.B, I do not currently recognize that group of identifiers because I am focused exclusively on LaTeX.

Table 1: How the default jdrlisting settings format different types of identifiers

basicstyle	identifier-style	LaTeX or 3rd-party emphasized identifiers	specially emphasized identifiers
1776[a]	nonsense[b]	equation[c]	jdrCodeSnip[d]

LaTeX control sequences	3rd-party control sequences	specially emphasized control sequences
\newcommand	\extrarowheight	\CodeSnipCharacterJdrLst[e]

```
% This is a comment string.
```

[a] Does not start with a letter (or \); thus not an identifier.

[b] An identifier (because all-alphabetic) but unrecognized as emphasis-worthy.

[c] A known (non–control sequence) identifier from either standard LaTeX or a 3rd-party package.

[d] A (non–control sequence) identifier from a package (viz., jdrlisting) being specially emphasized in this document.

[e] Control sequence from a package (viz., jdrlisting) being specially emphasized in this document.

The general syntax of \lstinline is:[23]

\lstinline[*key=value list*]*<char><verbatim code><char>* ⟨✂6⟩

For example, the code snippet:

```
Load the \lstinline|jdrlisting| package with \lstinline|\usepackage{jdrlisting}.
```
⟨✂7⟩

is typeset as:

Load the jdrlisting package with \usepackage{jdrlisting}.

Note that jdrlisting and \usepackage each receives special formatting. The listings package knows that \usepackage is a standard (i.e., built-in) TeX control sequence; the

[23] See section A.D.3 for more details on the syntax of \lstinline.

jdrlisting package knows that jdrlisting is a term of special interest in this documentation[24] (indeed, it's the package this document is focused on). For these reasons, each of these receives particular formatting. The particular colors are specified by the jdrlisting package.

Some special circumstances:

- If you want to include the verbatim code in a footnote, caption, sectioning command (e.g., \subsection{}), \mbox{}, or, more generally, in the argument of any command), you'll need to escape any of the following characters: {}#%\. See section IV.A.1 for details.
- If you're putting the verbatim code in the caption of a table or figure, and you also want to compile a \listoftables or \listoffigures, you'll need to do more. See section IV.A.1.2.
- If your \lstinline command is getting typeset at the end of a line (and there is additional text leftover to be typeset on the next line), you might benefit from wrapping the \lstinline command within an \mbox{}. See section IV.B.1.
- If you want to do something fancy and non-verbatim within the verbatim text you give to \lstinline, you might need to use the optional key [mathescape] when you call \lstinline.[25]

[[Note that I have inserted a \clearpage here solely to get this puppy to compile.]]

[24] See section VI.

[25] See ListingsDocs, § 4.14 ("Escaping to LaTeX") and, for examples, see infra sections V.F.3 and V.F.4.

III.E Typeset a snippet or one or a few lines of code set off and numbered like the equation environment using ⓐ the jdrCodeSnip environment in conjunction with ⓑ either the \lstinline command or the lstlisting environment

This package defines the jdrCodeSnip environment to display a snippet, or at most a few lines, of code, setting the code off from its previous code like a displayed equation—including providing a numeric label at the right margin to identify and reference it.

I break the possibilities down into:

- The snippet, when typeset, is shorter than a full line of the jdrCodeSnip environment. See section III.E.1.
- The single line of code, when typeset, is too long for a single line of the jdrCodeSnip environment. See section III.E.2.
- You want to display several lines of code in the jdrCodeSnip environment. See section III.E.3. In this case, you can choose either ⓐ multiple \lstinline commands (section III.E.3.1) or ⓑ a lstlisting environment (section III.E.3.2).

III.E.1 Snippet, when typeset, is shorter than a full line of the jdrCodeSnip environment: use \lstinline

If the snippet is shorter than a line, use the \lstinline command; for example, the following code:

```
\begin{jdrCodeSnip}
\label{codeeq:sampleShortLineCodeJdrCodeSnip}
\lstinline|\mbox{\lstinline$\\lstinline$}|
\end{jdrCodeSnip}
```

is typeset as this instance of the jdrCodeSnip environment:

```
\mbox{\lstinline$\\lstinline$}
```
⟨✂8⟩

III.E.2 Single line of code that is too long to be displayed on a single line of the jdrCodeSnip display: use multiple \lstinline commands

You might have a logically single line of code that is nevertheless, when typeset, too long to be displayed on a single line of the jdrCodeSnip environment's display. For example, consider the following single line of code:

```
\newcommand{\descriptorStylejdrLst}[1]{\textcolor{jdrDescriptorColor}{\textrm{\textit{#1}}}}
```

If you were to try to display this line of code using \lstinline and the jdrCodeSnip, you'd get:

```
\newcommand{\descriptorStylejdrLst}[1]{\textcolor{
jdrDescriptorColor}{\textrm{\textit{#1}}}}
```
⟨✂9⟩

which breaks arbitrarily at a nonoptimal point.

You can instead use *two* \lstinline commands within the jdrCodeSnip environment, choosing where you want to split the code; the first \lstinline carries the first part and the second \lstinline carries the remainder. You need to add an \\ after the first \lstinline (and, more generally, after all but the last \lstinline). You can add some leading spaces on the second line to provide indentation.[26]

For example, the following code snippet

```
\newcommand{\descriptorStylejdrLst}[1]
    {\textcolor{jdrDescriptorColor}{\textrm{\textit{#1}}}}
```
⟨✂10⟩

was produced by Listing 1.

Listing 1: Splitting a long-ish line of code into two separate lines

```
\begin{jdrCodeSnip}
%\label{codeeq:NAME}
\lstinline*\newcommand{\descriptorStylejdrLst}[1]* \\
\lstinline*    {\textcolor{jdrDescriptorColor}{\textrm{\textit{#1}}}}*
\end{jdrCodeSnip}
```

III.E.3 Display several lines of code in a jdrCodeSnip environment

You can display several lines of code in a jdrCodeSnip environment in either of two ways:

- with multiple \lstinline commands, one (or more) per line of code. See section III.E.3.1.
- with a lstlisting environment. See section III.E.3.2.

For the differences in the output of these two methods, see section III.E.3.3.

[26] See Werner's answer to "How to force \lstinline to add a line break," TEX Stack Exchange, April 4, 2016.

III.E.3.1 Use multiple \lstinline commands to display multiple lines of code within a jdrCodeSnip environment

You can similarly display several lines of code, using a separate \lstinline command for each line of code,[27] ending all but the last with \\. For example, as in:

\addListEmphClassJdrLst{*list*}{*n*}
\addListTeXCSClassJdrLst{*list*}{*n*} ⟨✂11⟩

which was produced with Listing 2.

Listing 2: Typesetting multiple lines of code with multiple \lstinline commands

```
\begin{jdrCodeSnip}
%\label{codeeq:NAME}
\lstinline[mathescape]|\addListEmphClassJdrLst{$\descriptorStylejdrLst{list}$}{$
    \descriptorStylejdrLst{n}$}|\\
\lstinline[mathescape]|\addListTeXCSClassJdrLst{$\descriptorStylejdrLst{list}$}{$
    \descriptorStylejdrLst{n}$}|
\end{jdrCodeSnip}
```

III.E.3.2 Use a lstlisting environment to display multiple lines of code within a jdrCodeSnip environment

Rather than using, within a jdrCodeSnip environment, multiple \lstinline commands to present multiple lines of code, you can use a lstlisting (or jdrLstListing[28]) environment to present the multiple lines of code.[29]

[27] If a single line of code is too long to comfortably and nonawkwardly display with a single \lstinline command, you can split that line of code up between two (or more) \lstinline commands as in section III.E.2.

[28] See section IV.C.2.2.

[29] The syntax of the lstlisting environment is discussed in section A.E.1.

For example, the output in code snippet ✂12 is produced using Listing 3.

```
\usepackage{listings}
\usepackage{xcolor}
\lstset{%
        basicstyle=\color{blue}\ttfamily,%
        keywordstyle=\color{red},%
        morekeywords={someKeyword}%
        }
```

⟨✂12⟩

When inside the jdrCodeSnip environment, do not use the float, caption, or label keys for the lstlisting environment. (There is no caption for a jdrCodeSnip environment. Its label is specified by jdrCodeSnip's own \label command.)

Listing 3: Example of lstlisting environment within jdrCodeSnip environment

```
\begin{jdrCodeSnip}
\label{codeeq:exampleLstlistingWithinJdrCodeSnip}
\begin{lstlisting}
\usepackage{listings}
\usepackage{xcolor}
\lstset{%
        basicstyle=\color{blue}\ttfamily,%
        keywordstyle=\color{red},%
        morekeywords={someKeyword}%
        }
\end{lstlisting}
\end{jdrCodeSnip}
```

III.E.3.3 Comparing using \lstinline vs. lstlisting within jdrCodeSnip

The differences between using \lstinline and using lstlisting within a jdrCodeSnip environment are summarized in Table 2.

Table 2: Comparing using \lstinline vs. lstlisting within jdrCodeSnip

Characteristic	\lstinline	lstlisting
Background color	None	colorBackgroundJdrLst
Text size[a]	Matches document font	\lstFontSizeDisplay
Line numbers?	No	Yes
Identifier	✂2	Listing 3

[a] See section XI.B for how this is accomplished.

III.E.4 Referencing a jdrCodeSnip environment by number; customizing the references

Each instance of the jdrCodeSnip environment is numbered near the right margin, e.g., ✂, similar to the way that an equation environment numbers equations.[30]

In order to reference the code snippet by its number, you need a pair of complementary commands. First, you need a \label command in the jdrCodeSnip environment itself, for example:[31]

```
\label{codeeq:exampleLstlistingWithinJdrCodeSnip}
```
⟨✂13⟩

which is the second line of the code in Listing 3.

Then, wherever you want to refer to that snippet, you use a corresponding \ref command:

```
\ref{codeeq:sampleShortLineCodeJdrCodeSnip}
```
⟨✂14⟩

which is typeset as "✂8."

[[Note that I have inserted a \clearpage here solely to get this puppy to compile.]]

30 The ✂ character can be customized. See section X.B.

31 There is nothing magic or mandatory about my suggestion of using codeeq as the beginning of the marker phrase. It is meant to be analogous to the common practice of, for example, using fig: as the beginning of a label for a figure or using tab: for the beginning of a label for a table. ("Since you can use exactly the same commands to reference almost anything, you might get a bit confused after you have introduced a lot of references. It is common practice among LaTeX users to add a few letters to the label to describe *what* you are referencing." LaTeX/Labels and Cross-referencing, Wikibooks.)

III.F Using the lstlisting environment as an optionally floating standalone environment in which to display longer sections of verbatim code

In section III.E.3.2, I showed how to use the lstlisting environment to display segments of code by wrapping the the lstlisting environment within a jdrCodeSnip environment, which labeled the segment (e.g., ✂12) in a way that could be referenced from elsewhere in a document.

The lstlisting environment[32] is sufficiently fully featured to use as a standalone environment for the display of longer code segments:

- you can give the listing a caption;
- the caption will show up in a list of Listings, if you choose to automatically create it;
- you can give the listing a label, and you can reference the listing from elsewhere in the document in a way that will look like, e.g., "Listing 3."[33]
- you can choose to float the listing, like a table or figure.

III.F.1 Simple no-frills example of lstlisting to display a code segment

Here I present a no-frills example of typesetting a segment of code, where by "no-frills" I mean no caption, no label, and it doesn't float. It just appears right where you place it.

To accomplish this, just wrap your code to typeset inside the following:

```
\begin{lstlisting}
%   Some code
\end{lstlisting}
```
⟨✂15⟩

This technique produces, for example, the following listing:

```
\definejdrsgame{4}{4}
\renewcommand{\Rplayernm}{}
\renewcommand{\Cplayernm}{}
%   Define Row strategies
\readarray{RowStrategies}{i&ii&iii&iv}
%   Define column's strategies
\readarray{ColumnStrategies}{I&II&III&IV}
\readarray{RowPayoffs}{9&0&0&0&0&9&0&0&0&0&10&0&0&0&0&9}
```

[32] The syntax of the lstlisting environment is discussed in section A.E.1.

[33] The command \lstlistingname, in this case "Listing", defines the text that prefixes the caption text. (ListingsDoc, § 4.9 on page 34.)

```
\readarray{ColumnPayoffs}{9&0&0&0&0&9&0&0&0&0&10&0&0&0&0&9}
\printjdrsgame
```

III.F.2 Add a caption and label to a lstlisting listing and/or to cause the listing to float

To add a caption and/or label to a lstlisting listing, and/or to cause the listing to float,[34] use the full general syntax for the lstlisting environment, which has an optional argument (that appears *after* the mandatory argument:

```
\begin{lstlisting}[key=value list]
%   Some code                                    ⟨✂<16⟩
\end{lstlisting}
```

Specifically:

- Use the float key to cause the listing to float;[35] You can assign float-placement directives, e.g., float=tp,[36], a subset of tbph.[37]
- Use the caption key to define a caption; assign the string to the key with a =, wrapping the string in {}; e.g., caption={SomeCaption}. As with the standard \caption command, you can specify an optional argument, for example for a shortened form of the caption to be used in the list of listings; e.g., {[*short*]*long*}.[38,39]
- Use the label key to define a label for the listing with which you can reference the

[34] You can have a caption or label without floating the listing. ("[H]ere it is also possible to have a caption regardless of whether or not the listing is in a float." ListingsDoc, § 2.7 on page 18. See also ListingsDoc, § 4.9 on page 34. "In despite of LaTeX standard behaviour, captions and flots are independent from each other here; you can use captions with non-floating listings.") It is not clear to me how/whether ⓐ not floating the listing is different from ⓑ floating the listing but specifying H as the position parameter (which requires, I believe, the float package.

[35] The presence of the float key is sufficient; you do not need to set it to true or any Boolean value with an = sign.

[36] ListingsDocs, § 1.4 on page 8. ("LaTeX's float mechanism allows one to determine the placement of floats. How can I do that with these? You can write 'float=tp', for example.")

[37] ListingsDoc, § 4.3 on page 28. "The argument controls where LaTeX is *allowed* to put the float: at the top or bottom of the current/next page, on a separate page, or here where the listing is."

[38] ListingsDoc, § 2.7 on page 18.

[39] Note carefully the syntax: the bounding encloses the optional argument [].

listing from elsewhere in the document; e.g., `label=lstlisting:NAME`.[40] (There is no need to wrap the label's name in `{}` as long as the label name doesn't contain a comma.)

- Make sure the *key=value list* pairs are comma separated.[41]

These rules are manifested in code snippet ✂17, which produces[42] the listing in Listing 4:

```
\begin{lstlisting}%
  [%
    float,%
    caption={Defining a new strategic-form game using the \lstinline|jdrsgame|
         package},%
    label=lstlisting:defineNewStrategicFormGame,%
  ]%
\renewcommand{\Rplayernm}{}
\renewcommand{\Cplayernm}{}
%   Define Row strategies
\readarray{RowStrategies}{i&ii&iii&iv}
%   Define column's strategies
\readarray{ColumnStrategies}{I&II&III&IV}
\readarray{RowPayoffs}{9&0&0&0&0&9&0&0&0&0&10&0&0&0&0&9}
\readarray{ColumnPayoffs}{9&0&0&0&0&9&0&0&0&0&10&0&0&0&0&9}
\printjdrsgame
\end{lstlisting}
```

⟨✂17⟩

40 There is nothing magic or mandatory about my suggestion of using `lstlisting` as the beginning of the marker phrase. It is meant to be analogous to the common practice of, for example, using `fig:` as the beginning of a label for a figure or using `tab:` for the beginning of a label for a table. ("Since you can use exactly the same commands to reference almost anything, you might get a bit confused after you have introduced a lot of references. It is common practice among LaTeX users to add a few letters to the label to describe *what* you are referencing." LaTeX/Labels and Cross-referencing, Wikibooks.)

41 Omitting a comma after any *key=value list* pair (except the last) will at least result in the following *key=value list* pairs being ignored.

42 *Except*, I actually produced Listing 4 with `jdrLstListing`. ⓐ This more-general syntax of `lstlisting` is a non-starter on Overleaf v1 because version 1 appears to unconditionally start treating everything after `\begin{lstlisting}` as verbatim without checking for an immediate `[` which signals that the optional argument is nonempty; however, the `lstlisting` formulation works fine on v1 if I omit the optional argument. ⓑ Under Overleaf v2, the code in ✂17 just times out. However, if I replace `lstlisting` with `jdrLstListing`, it compiles immediately. See Jim Ratliff, "Example: `lstlisting` fails to compile on Overleaf v2, but replacing `lstlisting` with `jdrLstListing` compiles immediately," Overleaf.

Listing 4: Defining a new strategic-form game using the jdrsgame package

```
\renewcommand{\Rplayernm}{}
\renewcommand{\Cplayernm}{}
%   Define Row strategies
\readarray{RowStrategies}{i&ii&iii&iv}
%   Define column's strategies
\readarray{ColumnStrategies}{I&II&III&IV}
\readarray{RowPayoffs}{9&0&0&0&0&9&0&0&0&0&10&0&0&0&0&9}
\readarray{ColumnPayoffs}{9&0&0&0&0&9&0&0&0&0&10&0&0&0&0&9}
\printjdrsgame
```

III.F.3 Create a list of Listings akin to the standard List of Tables and List of Figures

I explained in section III.F.2 how to add a caption to a standalone lstlisting environment (regardless of whether you float that environment). By default, such a listing gets an entry in the list of listings if one is created.

To print a list of listings, issue the command:[43]

```
\lstlistoflistings
```
⟨✂18⟩

[[Note that I have inserted a \clearpage here solely to get this puppy to compile.]]

[43] ListingsDoc, § 2.7 on page 18.

IV Some arcane features of listings you're likely to bump into (*ouch!*)

IV.A Certain characters ({}#%\) sometimes need to be escaped

There are some characters that sometimes need to be escaped by immediately preceding the character with a \.

This set of characters is:[44]

```
{}#%\
```
⟨✂19⟩

There are two contexts in which these characters must be escaped:

- When any of these characters appears in a *key*=*value* parameter list.[45]
- When a \lstinline command or a lstlisting environment appears in an argument to a command.[46]

I have never encountered the case where any of these characters appear in a *key*=*value* parameter list, so I won't consider this case further.

While there can be occasions for a lstlisting environment itself to be an argument of a command,[47] this is not a use case I have run into, so I do not consider it further.

Thus, I consider only the case in which a \lstinline command appears in an argument to a command. See section IV.A.1.

---

44 I note that this set of five characters is referenced in ListingsDocs, § 4.1 but only four of these (viz., excluding #) are referenced in ListingsDocs, § 5.1, in the discussion of a \lstinline command or a lstlisting environment appearing in an argument to a command. I don't know whether dropping the # from the list was a mistake or indicates that # is OK in an argument.

45 See ListingsDocs at § 4.1. ("Regarding the parameters… If you want to enter one of the special characters {}#%\, this character must be escaped with a backslash. This means that you must write \} for the single character 'right brace'—but of course not for the closing paramater character.")

46 See ListingsDocs at § 5.1. ("[I]f you want to use \lstinline or the listing environment inside arguments… *you* must work a bit more. You have to put a backslash in front of each of the following four characters: \{}%")

47 For example, ListingsDocs, § 5.1, gives the example of a lstlisting environment inside of an \fbox command.

IV.A.1 Escaping {}#%\ when a \lstinline command appears in an argument to a command

I noted in section IV.A that certain characters ({}#%\) need to be escaped when a \lstinline command appears in an argument to a command. In this section, I identify several contexts in which a \lstinline command would likely be used within an argument to a command and I give examples of how to properly escape the problematic characters.

Further, I identify a case—when \lstinline is in the argument to a \caption command for a table and you want to compile a \listoftables—where an additional step must be taken.[48]

There are several commonly encountered occasions on which \lstinline would show up in the argument to a command and, moreover, where one of the escapable characters would be involved.

Here are some parts of a document that are inside an argument to a command where you'd be likely to use \lstinline:[49]

- text within a \footnote{} command;
- text within a sectioning command, e.g., \subsection{};
- text within various commands in a tabular, such as \makecell or \multicolumn;
- text within an \mbox{};
- text within a \caption{} command;

In section IV.A.1.1, I present an example of a sentence that includes verbatim code that includes the problematic characters, and which is typeset by \lstinline. I show how to escape those problematic characters in order to place this sentence within the argument of a \footnote command.

---

48 See section IV.A.1.2.

49 I do not include the case of an item in an itemize environment, where each item appears to be in a sense an argument of a \item command (notwithstanding that the \item command does not take the standard delimiters). In my experience, there has been no need to escape any of the {}#%\ characters. Nevertheless, there are some posts that address the possiblility of problems with this combination: ⓐ "Using \lstinline inside a \item," TEX Stack Exchange, January 15, 2013, in which Martin Scharrer's answer contains a patch for \item; ⓑ "Using \lstinline inside an \item in beamer class in case of incremental overlay specifications?," TEX Stack Exchange, Denis Bitouzé; ⓒ "Problem in using \lstinline as description item," TEX Stack Exchange, June 6, 2015.

The escaping technique in section IV.A.1.1 is identical to the character escaping required for other commands, such as sectioning commands, \mbox commands, and within a \caption.

Section IV.B.1.2 provides an additional example of the required escaping of verboten characters in the context of the main use case in which you'd want to include a \lstinline expression within an \mbox.

In at least the case of a \lstinline command within the argument of a \caption command, it may not be enough just to escape the problematic characters. Specifically, if you also want to compile a List of Tables (using the \listoftables command) or a List of Figures (using the \listoffigures command), you should also take an additional step of defining an intermediate command using \DeclareRobustCommand. (See section IV.A.1.2.) Although this List of Tables and List of Figures use cases are the only scenarios that I've encountered where this extra step is necessary, there certainly may be other situations where it would solve problems.

IV.A.1.1 Example of escaping problematic characters in the verbatim code of a \lstinline command within an argument of another command

Consider the following text that appears *outside* an argument of a command:

> The syntax is \mycom{myopt}.

The corresponding LaTeX code is:

```
The syntax is \lstinline|\mycom{myopt}|.
```
⟨✂20⟩

If you want to move this sentence, and its LaTeX code, inside of an argument, e.g., of a footnote, sectioning command, caption, or \mbox, you need to transform code snippet ✂20 by escaping every instance of {}#%\ from within *the verbatim code* (and only the verbatim code). The transformed code is:

```
\footnote{The syntax is \lstinline|\\mycom\{myopt\}|}.
```
⟨✂21⟩

In code snippet ✂21, the following were escaped:

- The \ in \mycom
- The { in {myopt}
- The } in {myopt}

Note in particular that the \ in \lstinline was *not* escaped because \lstinline is not part of the verbatim code itself,[50] and it is only the verbatim code from which the problematic characters must be escaped.

IV.A.1.2 Within a \caption{} command, when you want to compile a \listoftables or \listoffigures, an extra step is needed: defining an intermediate macro with \DeclareRobustCommand

The case of using \lstinline within a \caption command within a table or figure environment poses additional problems if you also want to compile a List of Tables (using the \listoftables command) or List of Figures (using the \listoffigures command).[51]

Although the standard technique of escaping problematic characters (section IV.A.1.1) works to ensure that the caption itself is properly rendered, the presence of those problematic characters, even after being escaped, can either ⓐ prevent the List of Tables/Figures from compiling at all[52] or, in the alternative, ⓑ the escaped characters will not be rendered and the code with the escaped characters will not be properly highlighted.[53] For details of the problem I experienced, see the question I posed to TEX Stack Exchange.[54]

The solution, provided by David Carlisle,[55] is to first define a command, using \DeclareRobustCommand, to which is assigned the \lstinline command including its ver-

---

50 Of course, \lstinline could also be in verbatim code as in \lstinline|\lstinline|.

51 As far as I can tell, this problem does *not* infect captions for lstlisting environments. The \lstlistoflistings command creates a list of listings in which code with escaped characters is properly rendered.

52 In my experience, this was the result when I did not also load the caption package.

53 In my experience, this outcome occurred when I also loaded the caption package. Specifically, rather than seeing \newcommand, you'd see merely newcommand; i.e., the backslash would not be typeset and the identifier would not be recognized as a texcs identifier.

54 Jim Ratliff, "\listoftables problem: \lstinline and texcsstyle in \caption with and without caption package," TEX Stack Exchange, October 1, 2018.

55 See David Carlisle's answer to Jim Ratliff, "\listoftables problem: \lstinline and texcsstyle in \caption with and without caption package," TEX Stack Exchange, October 1, 2018.

batim code (with the problematic characters already escaped). This new command is then inserted within the argument of the \caption command.

I'll implement David's solution in the example of section IV.A.1.1. Let's start with the escaped version of the \lstinline command, excerpted from code snippet ✂21:

```
The syntax is \lstinline|\\mycom\{myopt\}|.
```
⟨✂22⟩

We now define a new command:

```
\DeclareRobustCommand{\mysyntax}%
    {\lstinline|\\mycom\{myopt\}|}
```
⟨✂23⟩

Note: It appears that this \DeclareRobustCommand command must be located *prior to* the \listoftables command (which is typically at the beginning of a document).[56]

Then, in the target \caption command, we replace the \lstinline command with \mysyntax:

```
\caption{The syntax is \mysyntax.}
```
⟨✂24⟩

The result is that the desired caption will appear properly highlighted both ⓐ as the caption of the table in the table environment itself and ⓑ as the caption associated with that table in the List of Tables.

See Listing 5 for a minimum working example of this approach. Its output is displayed in Figure 1.

[56] In other words, it appears that \listoftables needs to see the \DeclareRobustCommand command before \listoftables is encountered. This is surprising to me because my understanding is that lists, such as List of Tables, aren't compiled until a later pass through the document, in which case the \DeclareRobustCommand command will have been encountered no matter where it is located.

Listing 5: Minimum working example of using \DeclareRobustCommand to prepare content for a table caption

```
\documentclass{article}
\usepackage{listings,xcolor}
\lstset{language=[LaTeX]TeX,%
    basicstyle=\color{red}\ttfamily,%
    texcsstyle=*\color{green},%
    moretexcs={mycom},%
    }
\begin{document}
\DeclareRobustCommand{\mysyntax}{\lstinline|\\mycom\{myopt\}|}
\listoftables
\begin{table}[h]
    \caption{The syntax is \mysyntax}
    \begin{tabular}{|c|c|}
    \hline
       x  &  y\\
         \hline
    \end{tabular}
\end{table}
\end{document}
```

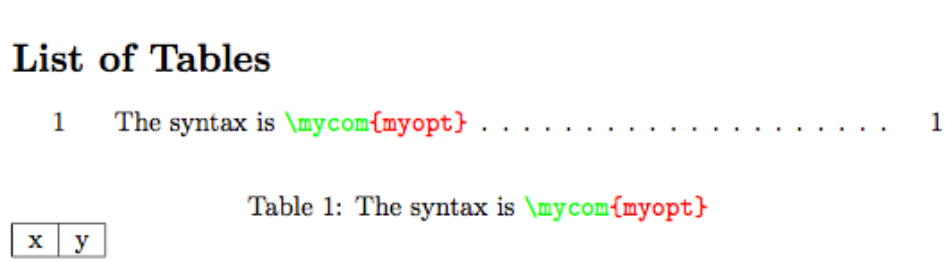

Figure 1: Output of the MWE from Listing 5

IV.B Wrapping \lstinline within an \mbox{} can be useful

IV.B.1 When a \lstinline is at the end of the line, you might need to wrap it within an \mbox

Suppose you have a \lstinline command that falls at the end of a typeset line, with more text in that paragraph awaiting to be typeset on the next line.

There are two situations in which this would create a typographical problem that you can solve through use of the \mbox command:

- LaTeX creates an unwanted "space" at the beginning of the next line. See section IV.B.1.1.
- If that \lstinline command is also the end of a sentence, and so is immediately followed by a period (.), LaTeX can choose to typeset the period alone on the next line, separated from the last word of the sentence. See section IV.B.1.2.

Both of these problems can be fixed by wrapping at least the `\lstinline` command itself within an `\mbox{}` command. Note, from the discussion in section IV.A.1, that any occurrences of `{}#%\` within the verbatim text of `\lstinline` must be escaped because the `\lstinline` command is within an argument of a command (viz., within the argument of `\mbox{}`).

IV.B.1.1 A `\lstinline` at the end of a typeset line can cause an unwanted space at the beginning of the next line

If you have a `\lstinline` command that lands at the end of a typeset line, with more text for that paragraph awaiting typesetting on the next line, it might create an unwanted white space at the beginning of the next line. For example, while drafting this very document I encountered the typographical snafu shown in Figure 2 where there is an unwanted leading "space" before "command" on the third line, which is caused by the `\lstinline|\lstinline|` command that winds up typeset as the last word on the previous line.

The jdrlisting package provides nine commands to facilitate document authors to format code identifiers that occur outside the verbatim contexts provided by the \lstinline command and the \lstlisting environment.[10] Eight of these commands permit the document author to match the formatting/highlighting that listings/jdrlisting pro-

Figure 2: Unwanted whitespace at beginning of a line caused by a `\lstinline` command at the end of the previous line

The LaTeX code for the offending sentence is:

```
The \lstinline$jdrlisting$ package provides nine commands to facilitate
    document authors to format code identifiers that occur outside the
    verbatim contexts provided by the \lstinline$\lstinline$ command and the
    \lstinline$\lstlisting$ environment.
```

⟨✂25⟩

This extra-leading-space problem is solved by wrapping `\lstinline$\lstinline$` within `\mbox{}` like so:

```
\mbox{\lstinline$\\lstinline$}
```

⟨✂26⟩

where note that I escaped the backslash from the verbatim `\lstinline` (i.e., the one wrapped in the `$` delimiters) but not from the actual command `\lstinline`. (This is an application of the escaping procedures discussed in greater detail in section IV.A.1.1.)

IV.B.1.2 If you end a sentence with a `\lstinline` snippet, you might need to wrap the snippet and its immediately following period inside an `\mbox`

If a normal sentence ends with a word immediately followed by a period, LaTeX will make sure that that period stays attached to the last letter of that last word. That last word may move to the next line, or may be hyphenated and its last syllable will move to the next line, but the period will definitely remain attached to the last letter. No way would LaTeX send the period alone to the next line.

Not so if you have a sentence that ends with a `\lstinline` command of verbatim code, which is then immediately followed by a period. LaTeX doesn't necessarily keep the period glued to the `\lstinline` command of verbatim code, and you can awkwardly have the period orphaned alone, beginning the next line.

If that happens, you can simply wrap both ⓐ the `\lstinline` command of verbatim code and ⓑ the immediately following period all within an `\mbox{}`. Do be aware that, because your `\lstinline` command of verbatim code is now appearing within the argument of a command, viz., the `\mbox`, you need to escape any occurrences of `{}#%\`.[57]

For example, suppose you have a sentence that ends: "after `\begin{document}`." The corresponding LaTeX code for this sentence fragment is:

```
after \lstinline|\begin{document}|.
```
⟨✂27⟩

Further suppose that, due to the particular length of the previous part of the paragraph, LaTeX orphans the period alone on the next line.

To cure this, you'd wrap an `\mbox{}` around both the `\lstinline|\begin{document}|` and the trailing period.

[57] See code snippet ✂19 and section IV.A.1.

A first cut at this would look like:

```
after \mbox{\lstinline|\begin{document}|.}
```
⟨✂28⟩

However, this would fail because the `\lstinline` expression ⓐ appears within the argument of a command (viz., `\mbox`) and ⓑ the verbatim code includes verboten characters `\`, `{`, and `}`. These verboten characters must be escaped from the verbatim code (and only from the verbatim code[58]) by prefixing each one with a `\`. The correct syntax is:[59]

```
after \mbox{\lstinline|\\begin\{document\}|.}
```
⟨✂29⟩

IV.B.2 Wrapping a `\lstinline` command within an `\mbox{}` can ease using the same content in ⓐ body text as ⓑ in footnotes, captions, or sectioning commands

While escaping `{}#%\` is necessary when the `\lstinline` appears in the argument to another command, it is also the case, conversely, that one should *not* escape these characters when the `\lstinline` does not appear within an argument to another command: Unnecessary escaping would yield the incorrect result in which the escaping `\`s were visible.

For example, if I wanted to display the string `{}#%\`, and I unnecessarily escaped each one, using the following:

```
\lstinline|\{\}\#\%\\|
```
⟨✂30⟩

the result would be: `\{\}\#\%\\`, rather than `{}#%\`.

This "feature"—that whether `{}#%\` should be escaped depends on whether the `\lstinline` command appears in the argument of another command—is unfortunate.

As a specific example, it's common to decide at some point in drafting to relegate text in the body to a footnote or, conversely, to later decide to elevate the prominence of footnote text by moving it to the body. Ideally, the document author should be able freely to copy-and-paste text between ⓐ the body and ⓑ a footnote, caption, or sectioning command.

---

58 To be clear, in code snippet ✂28, there's nothing objectionable about the `\` in `\lstinline`. The only `\` that needs escaping is the one in `\begin`.

59 This is an example of the same character-escaping technique explained in greater detail in section IV.A.1.1.

Instead—when the content includes a \lstinline command that includes {}#%\—① moving from body text to a footnote, say, involves escaping these characters and, conversely, ② moving from a footnote to body text requires *un*escaping these characters.

To the extent that this poses a problem to the document author, one solution is to ⓐ wrap all such constructions (\lstinline commands with {}#%\) in an \mbox{} from the get-go and ⓑ escape all the {}#%\ characters. Once wrapped inside an \mbox{}, these characters must be escaped regardless of whether the construction lies within body text or instead a footnote, caption, or sectioning command. Thus, the entire \mbox{} can be copied and pasted between these different contexts without changing the escaping of the {}#%\.

IV.C Using \lstinline or lstlisting to typeset code that itself includes \lstinline or lstlisting, respectively, code

The documentation you're reading this instant is an example of when you'd want to discuss code relating to the listings package. This documentation uses the listings package itself as the coding "language" to discuss itself, making the commands/environments of the listings package an embedded metalanguage for this purpose.[60]

IV.C.1 It's straightforward to use \lstinline to typeset code that includes \lstinline and any other listings entities

I discuss the syntax of the \lstinline command in section A.D.3:

\lstinline[*key=value list*]<*char*><*verbatim code*><*char*> ⟨✂31⟩

where <*char*> refers to a single character that is not found in <*verbatim code*>.[61] This single character both immediately precedes and immediately follows the string of code to be rendered verbatim.[62]

In effect, once LaTeX determines <*char*> by encountering it, LaTeX stops interpreting the subsequent characters as meeaningful LaTeX code until it encounters <*char*> a second time.

---

60 See, for example, "Types » Embedded" of "Metalanguage," Wikipedia, accessed October 2, 2018.

61 It's better to avoid using an opening square bracket (i.e., [) as the delimiter because \lstinline has to scan for that character because it would delimit the command's optional argument. See ListingsDocs at § 4.2. ("Since the command first looks ahead for an optional argument, you must provide at least an empty one if you want to use [as <*character*>.")

62 This convention, which leaves the delimiter undesignated ex ante, allows the delimiter to be chosen according to the string desired to be rendered verbatim.

As a result, there is no problem with the \lstinline command referencing itself, even deeply nested; this just requires changing the delimiting character for each level of reference:

```
\lstinline|\lstinline|
\lstinline*\lstinline|\lstinline|*
\lstinline^\lstinline*\lstinline|\lstinline|*^
```
⟨✂32⟩

The output of code snippet ✂32 is:

```
\lstinline
\lstinline|\lstinline|
\lstinline*\lstinline|\lstinline|*
```

IV.C.2 Choosing an environment to enclose code that includes \begin{lstlisting} and \end{lstlisting}

IV.C.2.1 The lstlisting environment cannot be nested because a verbatim \end{lstlisting} will cause early termination

Although it's simple for the \lstinline command to reference itself, because of its flexibly chosen delimiter character,[63] the same cannot be said for the lstlisting environment because its beginning and ending statements—\begin{lstlisting} and \end{lstlisting}, respectively—are fixed.

Consider the following attempt to nest the lstlisting environment:

```
\begin{lstlisting}
\begin{lstlisting}
% Here's some code
\end{lstlisting}
\end{lstlisting}
```
⟨✂33⟩

[63] See section IV.C.1.

The desired output from code snippet ✂33 would be:

```
\begin{lstlisting}
% Here's some code
\end{lstlisting}
```
⟨✂34⟩

However, the actual output from code snippet ✂33 is:

```
\begin{lstlisting}
% Here's some code
```
⟨✂35⟩

In other words, the output is cutoff prematurely—leaving `\end{lstlisting}` untypeset. This early termination occurs because the first time `\end{lstlisting}` is encountered, it is interpreted not as being part of the verbatim code to typeset but rather as the termination of the original `lstlisting` environment.[64,65]

IV.C.2.2 The `jdrLstListing` environment from the `jdrlisting` package can wrap a verbatim `lstlisting` environment

In section IV.C.2.1, I showed that you can't nest `lstlisting` environments because a verbatim instance of `\end{lstlisting}` is confused for being an actual instance of `\end{lstlisting}`, causing the environment to terminate prematurely and incompletely.

The `jdrlisting` package defines a new environment, `jdrLstListing`, to get around this problem.[66]

The `jdrLstListing` environment is nothing other than the `lstlisting` environment

---

64 This is also explained in Peter Grill's answer to "Masking `\end{lstlisting}`," TEXStack Exchange, October 18, 2011. ("I don't think you can nest within the `lstlisting` environment. Since this is a verbatim environment, the second `\begin{lstlisting}` is ignored, and the first `\end{lstlisting}` results in the termination of the outer `\begin{lstlisting}` leaving an extra `\end{lstlisting}`.")

65 Werner offers a workaround, using `[mathescape]` and a string `$$` between the `\` and `end{lstlisting}` of the verbatim `\end{lstlisting}`, in his answer to "Masking `\end{lstlisting}`," TEXStack Exchange, October 18, 2011.

66 This is suggested by Enrico Gregorio's answer to "Masking `\end{lstlisting}`," TEXStack Exchange, October 18, 2011. ("If you want to show examples of `lstlisting` itself, define a different environment [using `\lstnewenvironment`]....")

itself but with a different name. By invoking the jdrLstListing environment with \begin{jdrLstListing}, LaTeX will continue to treat all following text as verbatim, rather than meaningful LaTeX code, until an \end{jdrLstListing} is encountered. Thus, encountering a verbatim \end{lstlisting} will not cause the jdrLstListing environment environment to prematurely terminate.

We can transform the failed attempt to nest lstlisting environments, code snippet ✂33, by replacing the outer (actual, non-verbatim) lstlisting environment with an equivalent but differently named jdrLstListing environment:

```
\begin{jdrLstListing}
\begin{lstlisting}
% Here's some code
\end{lstlisting}
\end{jdrLstListing}
```

⟨✂36⟩

These reformed quasi-nested environments does indeed yield the desired output code snippet ✂34.

That said…although the jdrLstListing environment *theoretically* solves the problem (that lstlisting environments can't be nested), in practice the ceasefire appears fragile.[67]

IV.C.2.3 The LTXexample environment from the showexpl package is a verbatim alternative that simultaneously displays the output of the code

Rolf Niepraschk's showexpl package provides the LTXexample environment "to typeset LaTeX source code and the related result in the same document."[68]

[67] I've often had the experience where the incremental addition of additional verbatim code inexplicably breaks something and compilation fails. It has been impossible to track down through conventional divide-and-conquer methods what the problem is.

[68] For documentation, see Rolf Niepraschk, "The showexpl package," December 11, 2016. In addition, the author provides a GitHub repository and a document of examples as pdf and its source on GitHub.

The syntax of the LTXexample environment is:[69]

```
\begin{LTXexample}[key=value list]
%   Some code
\end{LTXexample}
```
⟨✂37⟩

where the *key=value list* allows the specification of any of more than a dozen parameters.[70] In particular, you can choose whether you want the output presented to the left, to the right, above, or below the source code.[71]

V Commands to emulate or extend listings syntax highlighting in text outside of \lstinline or lstlisting-type environments

Use of this package, jdrlisting, results in identifiers in listings being formatted according to the identifier's category (and class).

There are use cases in which a document author will want to be able to apply the same formatting to a term when it's not in a listing (i.e., neither an argument to \lstinline nor enclosed in a lstlisting environment).[72]

The commands listed in Table 3 and discussed below facilitate that.

V.A \typewriterBasicColorJdrLst to emulate basicstyle

The command \typewriterBasicColorJdrLst emulates the basicstyle format. Its syntax is:

```
\typewriterBasicColorJdrLst{term}
```
⟨✂38⟩

[69] Rolf Niepraschk, "The showexpl package," December 11, 2016, on pages 1–2.

[70] See Rolf Niepraschk, "The showexpl package," December 11, 2016, pages 1–2, for details. The document of examples (as pdf and its source on GitHub) has illustrations to illucidate the meanings of some of these parameters.

[71] As far as I can tell, there is no option to suppress the output entirely.

[72] [[This section would be strengthened by including examples, which could be drawn from elsewhere in this same document.]]

Table 3: Commands provided by the jdrlisting package to emulate or extend listings syntax highlighting in text outside of \lstinline or lstlisting-type environments, and where they are discussed in this document

Command	§	Comment
\typewriterBasicColorJdrLst	V.A	Emulate basicstyle
\typewriterIdentifierColorJdrLst	V.B	Emulate identifierstyle
\typewriterKeywordColorJdrLst	V.C	Emulate keywordstyle
\typewriterEmphColorJdrLstA	V.D	Emulate Class #1 of emphstyle
\typewriterEmphColorJdrLstB	V.D	Emulate Class #2 of emphstyle
\typewriterTeXCSColorJdrLstA	V.E	Emulate Class #1 of texcsstyle
\typewriterTeXCSColorJdrLstB	V.E	Emulate Class #2 of texcsstyle
\typewriterTeXCSColorJdrLstC	V.E	Emulate Class #3 of texcsstyle
\descriptorStylejdrLst	V.F	Format meta descriptors

V.B \typewriterIdentifierColorJdrLst to emulate identifierstyle

The command \typewriterIdentifierColorJdrLst emulates the identifierstyle format. Its syntax is:

\typewriterIdentifierColorJdrLst{*term*} ⟨✂39⟩

V.C \typewriterKeywordColorJdrLst to emulate keywordstyle

The command \typewriterKeywordColorJdrLst emulates the keywordstyle format. Its syntax is:

\typewriterKeywordColorJdrLst{*term*} ⟨✂40⟩

V.D \emphColorTypewriterjdrLstα commands to emulate the classes of emphstyle

There are two commands of the form \emphColorTypewriterjdrLstα, for α = A and B, that emulate Classes #1 and #2, respectively, of the emphstyle.

The command \typewriterEmphColorJdrLstA emulates class #1 of the emphstyle for-

mat. Its syntax is:

\typewriterEmphColorJdrLstA{*term*} ⟨✂41⟩

The command \typewriterEmphColorJdrLstB emulates class #2 of the emphstyle format. Its syntax is:

\typewriterEmphColorJdrLstB{*term*} ⟨✂42⟩

V.E \TeXCSColorTypewriterjdrLstα commands to emulate the classes of texcsstyle

There are two commands of the form \TeXCSColorTypewriterjdrLstα, for α = A, B, and C, that emulate Classes #1, #2, and #3, respectively, of the texcsstyle.

The command \typewriterTeXCSColorJdrLstA emulates class #1 of the texcsstyle format. Its syntax is:

\typewriterTeXCSColorJdrLstA{*term*} ⟨✂43⟩

The command \typewriterTeXCSColorJdrLstB emulates class #2 of the texcsstyle format. Its syntax is:

\typewriterTeXCSColorJdrLstB{*term*} ⟨✂44⟩

The command \typewriterTeXCSColorJdrLstC emulates class #3 of the texcsstyle format. Its syntax is:

\typewriterTeXCSColorJdrLstC{*term*} ⟨✂45⟩

V.F \descriptorStylejdrLst to format meta descriptors

V.F.1 A meta descriptor can explain what an argument is expecting

It's best to start this discussion with an example. Suppose we want to discuss the syntax of a command \jdrHlineTC, which takes two arguments. Part of the syntax is specifying what kinds of constant or variable should be put into each of the arguments. We could

describe this syntax as:

```
\jdrHlineTC{length}{color}
```
⟨✂46⟩

In code snippet ✂46, *length* and *color* are what I will call *descriptors*. Each appears between curly braces where an argument would appear in actual code.

The *descriptor* is a category of word or phrase that ⓐ you would embed within a code snippet but ⓑ is not literally code itself—you wouldn't actually find it in code; it might be quite syntactically incorrect if you did. Moreover, a descriptor is not a type of identifier that is recognized by the `listings` package.

A descriptor is *metacode*: it is useful for *talking about code*. In particular, the descriptor's purpose is to indicate to the reader what type of code entity should be inserted in lieu of the descriptor and what that code entity's interpretation is.

You'd never literally write `length` where *length* appears. Instead, you'd supply a valid expression for a length, e.g., `10pt` or `\arrayrulewidth`. Likewise, you'd never write `color` where *color* appears; you'd instead supply a defined name that refers to a valid color, e.g., `MidnightBlue`.

V.F.2 The `\descriptorStylejdrLst` command applies descriptor-specific formatting to its argument

Because a descriptor is not typically found in code, it is less likely that ⓐ you'd want to typeset a descriptor in a `lstlisting` environment than that ⓑ you'd want to typeset a descriptor in inline code using `\lstinline`.

Because descriptors are not recognized by the `listings` package as a known type of snippet, that package does not automatically specially format descriptors.

To allow for appropriate syntax highlighting for descriptors, the `jdrlisting` package defines a command, `\descriptorStylejdrLst`, that applies the desired formatting to its argument. This command is defined to be:

```
\newcommand{\descriptorStylejdrLst}[1]%
    {\textcolor{jdrDescriptorColor}{\textrm{\textit{#1}}}}
```
⟨✂47⟩

i.e., it sets the type to ⓐ a serif font that ⓑ is italicized and ⓒ colored as colorDescriptorJdrLst, e.g., *some descriptor*.

V.F.3 Using \descriptorStylejdrLst within \lstinline requires the optional [mathescape] key in order to "escape to LaTeX"

In code snippet ✂46, the descriptor portions (viz., *length* and *color*) appear embedded within verbatim code formatted by \lstinline.

To accomplish that we need to tell \lstinline to pause its verbatim approach to its argument and instead, temporarily, interpret \descriptorStylejdrLst{length} and \descriptorStylejdrLst{color} as actual LaTeX instructions to process. This is called "escaping to LaTeX." To do this, we use the optional key [mathescape] when we call \lstinline.[73] mathescape, when present or true, "activates… special behavior of the dollar sign. If activated a dollar sign acts at TeX's text math shift."

Using this capability, code snippet ✂46 is achieved by the following \lstinline command:

```
\lstinline[mathescape]|%
    \jdrHlineTC{$\descriptorStylejdrLst{length}$}%
    {$\descriptorStylejdrLst{color}$}|
```
⟨✂48⟩

V.F.4 Using math mode within \descriptorStylejdrLst to achieve, e.g., n_{args}, in conjunction with \lstinline[mathescape]

You might want to use \descriptorStylejdrLst to produce a descriptor that is truly a math-mode expression, e.g., n_{args}.[74]

If you are using \descriptorStylejdrLst outside of the \lstinline[mathescape] context, this is accomplished as simply as wrapping the desired math-mode expression between $…$:

```
\descriptorStylejdrLst{$n_{args}$}
```
⟨✂49⟩

However, *inside* the \lstinline[mathescape] context, this attempt, i.e.,

```
\lstinline[mathescape]|$\descriptorStylejdrLst{$n_{args}$}$|
```
⟨✂50⟩

[73] See ListingsDocs, § 4.14 ("Escaping to LaTeX").

[74] See, for example, code snippet ✂82 in section A.D.5.

fails because the opening `$` of `$n_{args}$` is perceived to be the closing `$` of `\lstinline[mathescape]$`.

The workaround is to "hide" the `$`s of `$n_{args}$` by first defining a command:

```
\newcommand{\nArgs}{$n_{args}$}
```
⟨✂51⟩

and then insert `\nArgs` into the argument of `\descriptorStylejdrLst`:

```
\lstinline[mathescape]|$\descriptorStylejdrLst{\nArgs}$|
```
⟨✂52⟩

The workaround is incomplete, however. I haven't found a way to achieve n_{args} where the "args" appears as non–math mode text, i.e., as n_{args}. And this is true even outside the `\lstinline[mathescape]` context. For example, in `\descriptorStylejdrLst{$n_\text{args}$}`, the `\text{}` is apparently ignored or overridden; this renders as n_{args}.

VI Specially emphasizing the identifiers from a particular package when writing documentation about that package

Section VII lays out the scheme I have chosen that specifies how:

- TEX control sequences are divided between three classes of `texcs` identifiers:
  - ◇ Class #1: TEX control sequences in the `listings` definition of LATEX
  - ◇ Class #2: TEX control sequences harvested from third-party packages (including, by default, my own packages)
  - ◇ Class #3: TEX control sequences of particular document-specific significance
- identifiers—excluding TEX control sequences—such as names of packages, environments, keys, and values are divided between two classes of `emph` identifiers
  - ◇ Class #1: Names of packages, environments, keys, and values from standard LATEX or third-party packages, including by default my own packages
  - ◇ Class #2: Names of packages, environments, keys, and values from packages of particular significance to the specific document

In other words, by default the document-specific classes of both types of identifiers, viz., ⓐ Class #3 of the `texcs` identifiers and ⓑ Class #2 of the `emph` identifiers, are empty.

By default, all identifiers from third-party packages reside either in ⓐ Class #2 of texcs identifiers or ⓑ Class #1 of emph identifiers.

In order to have the identifiers associated with a particular package be specially highlighted to make them distinctly highlighted from identifiers from standard LaTeX or other third-party packages, the document author must:

- promote the TeX control sequences of the distinguished package from Class #2 to Class #3 of texcs identifiers, and
- promote the emph identifiers of the distinguished package from Class #1 to Class #2 of emph identifiers.

This jdrlisting package provides a set of commands such that a document author can specially highlight both types of identifiers for a given package by issuing a single command.

These commands are of the form \emphasizeIdentifiers*x*, where *x* identifies the particular package. For example, when *x* = JdrArticle, the command is \emphasizeIdentifiersJdrArticle, which emphasizes the identifiers for the jdrarticle package.

These commands are listed and described in Table 4.

Table 4: Commands provided by the jdrlisting package to emphasize/unemphasize the identifiers of particular JDR packages, and where they are discussed in this document

Command	Description
\emphasizeIdentifiersJdrListing	Emphasize identifiers from the jdrlisting package
\unemphasizeIdentifiersJdrListing	Unemphasize identifiers from the jdrlisting package
\emphasizeIdentifiersJdrSgame	Emphasize identifiers from the jdrsgame package
\unemphasizeIdentifiersJdrSgame	Unemphasize identifiers from the jdrsgame package
\emphasizeIdentifiersJdrUnicode	Emphasize identifiers from the jdrunicode package
\unemphasizeIdentifiersJdrUnicode	Unemphasize identifiers from the jdrunicode package
\emphasizeIdentifiersJdrArticle	Emphasize identifiers from the jdrarticle package
\unemphasizeIdentifiersJdrArticle	Unemphasize identifiers from the jdrarticle package
\emphasizeIdentifiersJdrHcline	Emphasize identifiers from the jdrhcline package
\unemphasizeIdentifiersJdrHcline	Unemphasize identifiers from the jdrhcline package

VII My scheme of identifier-category classes and what identifiers to assign to each of them

The remainder of this documentation assumes that you're familiar with the discussion of the listings package in section A.

The jdrlisting package embodies my personalized scheme for syntax highlighting LaTeX code in conjunction with the listings package. The most fundamental characteristics of that scheme are how to group identifiers and how, on that basis, to format them. Table 5 details my scheme.

VII.A TeX control sequences

I implement three classes of the texcs identifier category. See section VII.A.1, section VII.A.2, and section VII.A.3, respectively.

Each of the three classes receives a color with a name of the form jdrLstColorTeXCSStyleα, where α is A, B, or C, respectively.

VII.A.1 Class #1: TEX control sequences in the listings definition of LATEX

The first class is for those TEX control sequences that are part of the definition within listings of the LATEX dialect of the TEX language.[75]

Implementing this class *should* require no effort on the part of the jdrlisting package, because it should be preconfigured by the listings package. However, some standard-LATEX control sequences are defined in the language specification in such a way that they are not actually recognized as texcs;[76] I refer to these as omitted TEX control sequences. I added a subset of these omitted TEX control sequences to a definition in jdrlisting that assigns them to this first class of texcs identifiers.[77,78]

This class of language/dialect–defined TEX control sequences receives the color named colorTeXCSStyleJdrLstA.

---

75 See Carsten Heinz et al. "Language, Style and Format drivers for listings," September 2, 2018, version 1.7.

76 These control sequences *are* declared as values for the moretexcs key, but in a command \lst@definelanguage[AlLaTeX]{TeX}[LaTeX]{TeX}; whereas the other, effective, texcs declarations are in a command \lst@definelanguage[LaTeX]{TeX}[common]{TeX}.

77 The subset is added to the definition of the command \moreTeXcsOmittedStandardLaTeX.

78 The following is a list of the omitted TEX control sequences; the subset that I manually added to the first class are the control sequences (that are colored like this): \AtBeginDocument, \AtBeginDvi, \AtEndDocument, \AtEndOfClass, \AtEndOfPackage, \ClassError, \ClassInfo, \ClassWarning, \ClassWarningNoLine, \CurrentOption, \DeclareErrorFont, \DeclareFixedFont, \DeclareFontEncoding, \DeclareFontEncodingDefaults, \DeclareFontFamily, \DeclareFontShape, \DeclareFontSubstitution, \DeclareMathAccent, \DeclareMathAlphabet, \DeclareMathAlphabet, \DeclareMathDelimiter, \DeclareMathRadical, \DeclareMathSizes, \DeclareMathSymbol, \DeclareMathVersion, \DeclareOldFontCommand, \DeclareOption, \DeclarePreloadSizes, \DeclareRobustCommand, \DeclareSizeFunction, \DeclareSymbolFont, \DeclareSymbolFontAlphabet, \DeclareTextAccent, \DeclareTextAccentDefault, \DeclareTextCommand, \DeclareTextCommandDefault, \DeclareTextComposite, \DeclareTextCompositeCommand, \DeclareTextFontCommand, \DeclareTextSymbol, \DeclareTextSymbolDefault, \ExecuteOptions, \GenericError, \GenericInfo, \GenericWarning, \IfFileExists, \InputIfFileExists, \LoadClass, \LoadClassWithOptions, \MessageBreak, \OptionNotUsed, \PackageError, \PackageInfo, \PackageWarning, \PackageWarningNoLine, \PassOptionsToClass, \PassOptionsToPackage, \ProcessOptionsProvidesClass, \ProvidesFile, \ProvidesFile, \ProvidesPackage, \ProvideTextCommand, \RequirePackage, \RequirePackageWithOptions, \SetMathAlphabet, \SetSymbolFont, \TextSymbolUnavailable, \UseTextAccent, \UseTextSymbol.)

VII.A.2 Class #2: TEX control sequences harvested from third-party packages

The second class is additional TEX control sequences that I have harvested from third-party packages and manually added.[79]

This class is implemented wholly within the jdrlisting package. I expect it to grow over time as new commands are referenced in documents. This class, and its contents, will be made available in the default configuration of loading jdrlisting.

This class of third-party TEX control sequences receives the color named colorTeXCSStyleJdrLstB.

VII.A.3 Class #3: TEX control sequences of particular document-specific significance

The third class is TEX control sequences that are of particular document-specific interest—for example, commands from a package being discussed in that document—and thus might warrant distinct highlighting.

Identifiers in the this class can be added by the document author from within that document, using the appropriate moreemph key.

I may also bundle sets of these identifiers in jdrlisting so that the document author can issue a command and optionally assign all the identifiers in a bundle to the author's choice of Class #2 or Class #3.[80]

This class of document-specific TEX control sequences receives the color named colorTeXCSStyleJdrLstC.

[79] This harvesting has not been exhaustive.

[80] In this way, the author of a document can load such a bundle, e.g., identifiers from the jdrsgame package ⓐ into Class #2 for a document that displays code that uses jdrsgame but where jdrsgame is not a focus or, alternatively, ⓑ into Class #3 for a document that discusses use of the jdrsgame package as a prime focus.

Table 5: My scheme of LaTeX-specific category classes and what to assign to each of them

Code category	Class	What it contains	Color suffix jdrLstColor-
texcs	1	TeX control sequences defined by the language[a]	TeXCSStyleA
texcs	2	Third-party control sequences	TeXCSStyleB
texcs	3	Control sequences of particular interest	TeXCSStyleC
Keywords	—	—	—
Emphasized identifiers	1	Names of packages, environments, keys, and values from standard LaTeX or third-party packages	EmpStyleA
Emphasized identifiers	2	Names of packages, environments, keys, and values of particular interest	EmphStyleB

[a] See Carsten Heinz et al. "Language, Style and Format drivers for listings," September 2, 2018, version 1.7, § 2.79.

VII.B Identifiers to emphasize other than control sequences: names of packages, environments, keys, and values

In addition to TeX control sequences, I also want to be able to emphasize certain other identifiers. I consider as a single group all non–control sequences that are names of packages, environments, keys, or defined values.[81]

I divide these into two classes (that roughly correspond to the second and third classes of TeX control sequences discussed in section VII.A).

In my review of the definition of LaTeX, I found only TeX control sequences but no other identifiers.[82] Thus there are no language/dialect-defined names that would be analogous to the first class of TeX control sequences in the sense of being identified within the

[81] I considered, but rejected, breaking these identifiers into several groups: ⓐ packages and environments, ⓑ keys, and ⓒ values. I ultimately decided that doing so would be overkill.

[82] It appears that there *was* an (ineffective) attempt in the language definition of listings to define keywords for LaTeX. The cause of the ineffectiveness of that effort is similar to the problem that arose with "omitted" TeX control sequences. (See section VII.A.1.) In particular, these keywords *are* declared as values for the morekeywords key, but in a command \lst@definelanguage[AlLaTeX]{TeX}[LaTeX]{TeX}; whereas the other, effective, texcs declarations are in a command \lst@definelanguage[LaTeX]{TeX}[common]{TeX}. The initially "omitted keywords" are array, center, displaymath, document, enumerate, eqnarray, equation, flushleft, flushright, itemize, list, lrbox, math, minipage, picture, sloppypar, tabbing, tabular, trivlist, verbatim. I have manually assigned them to the first class of non–control sequence identifiers by including them in the definition of the command \moreEmphOmittedStandardLaTeX. See section IX.B.

listings package's definition of the LaTeX dialect of TeX. Though apparently not part of that language definition, there are names of at least environments that belong to standard LaTeX that I want to include, so I am not limiting the scope to third-party packages.

Each of the two classes receives a color with a name of the form jdrLstColorEmphStyleα, where α is A or B, respectively.

VII.B.1 Class #1: Names of packages, environments, keys, and values from standard LaTeX or third-party packages

The first class of non–control sequence identifiers is the set of identifiers that are names of either a package, environment, key, or defined value as found either in standard LaTeX or in a third-party package.

This class is implemented wholly within the jdrlisting package. I expect it to grow over time as new commands are referenced in documents. This class, and its contents, will be made available in the default configuration of loading jdrlisting.

This class of third-party TeX control sequences receives the color named colorEmphStyleJdrLstA.

VII.B.2 Class #2: Names of packages, environments, keys, and values from packages of particular significance to the specific document

The third class of non–control sequence identifiers is the set of identifiers that are names of either a package, environment, key, or defined value that is of particular document-specific interest—for example, that arise from a package being discussed in that document—and thus might warrant distinct highlighting.

Identifiers in the this class can be added by the document author from within that document, using the appropriate moreemph key.

I may also bundle sets of these identifiers in jdrlisting so that the document author can issue a command and optionally assign all the identifiers in a bundle to the author's choice of Class #1 or Class #1.[83]

This class of document-specific TeX control sequences receives the color named colorEmphStyleJdrLstB.

---

[83] In this way, the author of a document can load such a bundle, e.g., identifiers from the jdrsgame package ⓐ into Class #1 for a document that displays code that uses jdrsgame but where jdrsgame is not a focus or, alternatively, ⓑ into Class #2 for a document that discusses use of the jdrsgame package as a prime focus.

VIII Commands to add or delete a list of identifiers to a class or to move those identifiers between classes

The jdrlisting package defines six commands to facilitate adding (§ VIII.A) or deleting (§ VIII.B) a list of identifiers to a particular class or moving those identifiers between classes (§ VIII.C).

These commands are listed and described in Table 6 and discussed below.[84]

Table 6: Commands provided by the jdrlisting package to add or delete a list of identifiers to a class or to move those identifiers between classes, and where they are discussed in this document

Command	§	Comment
\addListEmphClassJdrLst	VIII.A	Add identifiers to an emph class
\addListTeXCSClassJdrLst	VIII.A	Add identifiers to a texcs class
\deleteListEmphClassJdrLst	VIII.B	Delete identifiers to an emph class
\deleteListTeXCSClassJdrLst	VIII.B	Delete identifiers to a texcs class yaba
\moveListEmphAtoBJdrLst	VIII.C	Move identifiers between two emph classes
\moveListTeXCSAtoBJdrLst	VIII.C	Move identifiers between two texcs classes

VIII.A Adding a list of identifiers to a class with \addListEmphClassJdrLst and \addListTeXCSClassJdrLst

You can add a list of identifiers to a particular class of ⓐ emphasized identifiers (i.e., an emph class) using the \addListEmphClassJdrLst command or ⓑ TEX control sequences using \addListTeXCSClassJdrLst. Their respective syntaxes are:

```
\addListEmphClassJdrLst{list}{n}
\addListTeXCSClassJdrLst{list}{n}
```
⟨✂53⟩

where

[84] The key hints that allowed me to solve this were ⓐ David Carlisle's answer to "Forcing macro expansion with keyval," TEX Stack Exchange, February 26, 2013, and ⓑ Joseph Wright's answer to "Building keyval arguments using a macro," TEX Stack Exchange, March 15, 2011. Also related is "How to define macros in order to reuse key-value parameters?," TEX Stack Exchange, February 8, 2015.

- *list* is a comma-separated list of identifiers you want to add to a particular class (of ⓐ emphasized identifiers or ⓑ TEX control sequences, respectively). *list* can also be a LATEX command that expands to a comma-separated list of identifiers.
- *n* is a natural number that identifies the number of the class to which the identifiers in *list* should be added. *n* can also be a LATEX command that expands to such a natural number.

VIII.B Deleting a list of identifiers to a class with \deleteListEmphClassJdrLst and \deleteListTeXCSClassJdrLst

You can delete a list of identifiers from a particular class of ⓐ emphasized identifiers (i.e., an emph class) using the \deleteListEmphClassJdrLst command or ⓑ TEX control sequences using the \deleteListTeXCSClassJdrLst command. Their respective syntaxes are:

```
\deleteListEmphClassJdrLst{list}{n}
\deleteListTeXCSClassJdrLst{list}{n}
```
⟨✂54⟩

where

- *list* is a comma-separated list of identifiers you want to delete from a particular class (of ⓐ emphasized identifiers or ⓑ TEX control sequences, respectively). *list* can also be a LATEX command that expands to a comma-separated list of identifiers.
- *n* is a natural number that identifies the number of the class of emphasized identifiers from which the identifiers in *list* should be deleted. *n* can also be a LATEX command that expands to such a natural number.

VIII.C Moving a list of identifiers from one class to another with \moveListEmphAtoBJdrLst and \moveListTeXCSAtoBJdrLst

You can move a list of identifiers from one particular class of ⓐ emphasized identifiers (i.e., an emph class) to another class of emphasized identifiers using the \moveListEmphAtoBJdrLst command or ⓑ TEX control sequences using the

\moveListTeXCSAtoBJdrLst command.[85] Their respective syntaxes are:

```
\moveListEmphAtoBJdrLst{list}{m}{n}
\moveListTeXCSAtoBJdrLst{list}{m}{n}
```
⟨✂55⟩

where

- *list* is a comma-separated list of identifiers you want to move between either ⓐ two classes of emphasized identifiers or ⓑ two classes of TEX control sequences. *list* can also be a LATEX command that expands to a comma-separated list of identifiers.
- *m* is a natural number that identifies the number of the class from which the identifiers in *list* should be moved. *n* can also be a LATEX command that expands to such a natural number.
- *n* is a natural number that identifies the number of the class to which the identifiers in *list* should be moved. *n* can also be a LATEX command that expands to such a natural number.

IX Managing identifiers

IX.A Defining a command as a comma-separated list of similarly situated identifiers

It is useful to define lists of identifiers that are similarly situated in the sense that they would share a common format.

For example, such lists make possible the technique discussed in section V, whereby all the identifiers from a particular package could receive privileged formatting in a document where that package was of special focus, but receive typical formatting in other documents. This technique is made possible by the set of six commands (e.g., \addListEmphClassJdrLst) that I discussed in section VIII, each of which operates on a comma-separated list of identifiers and either adds those identifiers to a particular class of texcs or emph identifiers, deletes those identifiers from such a class, or moves those identifiers between two classes of the same type of identifier.[86]

---

85 In other words, this is equivalent to ⓐ deleting the list from one class using the \deleteListEmphClassJdrLst (respectively, \deleteListTeXCSClassJdrLst) command and then ⓑ adding the same list to the other class using the \addListEmphClassJdrLst (respectively, \addListTeXCSClassJdrLst) command. The \moveListEmphAtoBJdrLst command simply combines these two steps into one.

86 See also Table 6 for the list of these commands.

IX.B Populating the identifiers for each style using one or more commands representing comma-separated lists of identifiers

There are three types of styles that control the highlighting of specified identifiers: texcsstyle, emphstyle and keywordstyle.[87] Each of these can spawn additional styles by the definition of additional classes.[88] Because I am focused exclusively on LaTeX code, I do not currently use keywordstyle in any way.

Defining how all identifiers are to be formatted requires assigning identifiers to each class of these three types of styles using the keys moretexcs, moreemph, and morekeywords.[89]

I have identified (so far) twelve sets of identifiers such that the identifiers within each set should share a common formatting. These twelve sets are distinguished from one another on one or both of two dimesions: ① whether they are texcs identifiers or emph identifiers and ② whether they are ⓐ standard LaTeX, ⓑ a package from a third-party other than Jim Ratliff, or ⓒ which Jim Ratliff package they are from.

For each of these twelve sets, I define a LaTeX command to store the identifiers in that set. For example, \moreTeXcsNonJdrThirdPartyLaTeX is the command to which is assigned all the texcs identifiers from packages by third parties other than Jim Ratliff.

The commands associated with these twelve sets are shown in Table 7.

87 There are other styles but they do not, except as a fall-back style, govern the formatting of specified identifiers. See Table 6.

88 See section A.B.2 and Table 12.

89 See section A.C.4.

Table 7: Commands provided by the jdrlisting package each of which stores a list of identifiers for a particular class of emph or texcs identifiers

Command	Type of identifier	Class	Comment
\moreTeXcsOmittedStandardLaTeX	texcs	1	Omitted from listings definition for LaTeX
\moreEmphOmittedStandardLaTeX	emph	1	Omitted from listings definition for LaTeX
\moreTeXcsNonJdrThirdPartyLaTeX	texcs	1	Non-JDR 3rd-party packages
\moreEmphNonJdrThirdParty	emph	1	Non-JDR 3rd-party packages
\JdrSgameMoreTeXcsJdrLst	texcs	1	jdrsgame package
\JdrSgameMoreEmphJdrLst	emph	1	jdrsgame package
\JdrUnicodeMoreTeXcsJdrLst	texcs	1	jdrunicode package
\JdrUnicodeMoreEmphJdrLst	emph	1	jdrunicode package
\JdrHclineMoreTeXcsJdrLst	texcs	1	jdrhcline package
\JdrUnicodeMoreEmphJdrLst	emph	1	jdrhcline package
\JdrArticleMoreTeXcsJdrLst	texcs	1	jdrarticle package
\JdrArticleMoreEmphJdrLst	emph	1	jdrarticle package

IX.C Managing the conflict when an identifier is both the name of a LaTeX command and a non-command identifier

There are cases where the same term, say foo, will have dual use as ⓐ a LaTeX command sequence, \foo, as well as ⓑ a non-command identifier, foo, that serves as the name of a package, a key, or a defined value. Examples include:

- \emph is a command to emphasize text by toggling italics;[90] But emph is also the name of a key in the same jdrlisting package discussed in this documentation.[91]
- In particular, sometimes a package has the same name as a command that that package defines. That's the case for the command \makecell, which is defined by the makecell package.

[90] See, e.g., "Bold, italics and underlining," Overleaf.

[91] See section A.B.2.3.

- \caption is a TEX control sequence. caption is also be a key in the listings package, e.g., \begin{lstlisting}[float,caption=A floating example].

Unfortunately, there is a conflict that prevents the same term foo being simultaneously recognized as ⓐ the command \foo and ⓑ the non-command emphasized identifier foo.[92] If a term foo exists as both ⓐ a TEX control sequence via moretexcs and ⓑ an emphasized identifier via emph, the control sequence \foo will be formatted as \emphstyle.[93]

To see this, consider the following example, where the LATEX code is on the right and its corresponding output is on the left. All three control sequences \myControlSequenceA, \myControlSequenceB, and \myControlSequenceC are declared as such using the key moretexcs. In addition, the base name for each of two of those commands is also defined as either an emphasized identifier (\\myControlSequenceA), a keyword (\\myControlSequenceB).

Because \myControlSequenceA, \myControlSequenceB, and \myControlSequenceC are each declared to be TEX control sequences, they should be rendered in green according to texcsstyle=*\color{green}. However, only \myControlSequenceC is properly colored, while the other two control sequences wear the colors of emphstyle and keywordstyle, respectively. The failure of \myControlSequenceA and \myControlSequenceB to be properly highlighted in green is due to the fact that each of their base names is also defined to be a non-command identifier.

[92] I have not tested whether the same conflict arises if you define a term to be both a TEX control sequence and as a keyword.

[93] "Bug: texcs…interferes with other keyword lists. If, for example, emph contains the word foo, then the control sequence \foo will show up in emphstyle." (ListingsDocs, § 4.6 on page 31.)

```
\myControlSequenceA
\myControlSequenceB
\myControlSequenceC
myControlSequenceA
myControlSequenceB
```

```
\documentclass{article}
\usepackage{listings}
\usepackage{xcolor}
\lstset{%
    language=[LaTeX]TeX,%
    basicstyle=\ttfamily,%
    emphstyle=\color{red},%
    keywordstyle=\color{blue},%
    texcsstyle=*\color{green},%
    moretexcs={myControlSequenceA,
        myControlSequenceB,myControlSequenceC},%
    moreemph={myControlSequenceA},%
    morekeywords={myControlSequenceB}%
}
\begin{document}
\begin{lstlisting}
\myControlSequenceA
\myControlSequenceB
\myControlSequenceC
myControlSequenceA
myControlSequenceB
\end{lstlisting}
\end{document}
```

Thus, for any dual-use base name `foo`, `\foo` and `foo` cannot both simultaneously be recognized for the command and non-command identifier, respectively, they are. That means that you have to decide whether you want the command flavor to be highlighted or the non-command flavor to be highlighted.

Fortunately, such a decision is not irrevocable. Indeed, you can toggle the interpretation of a base name back and forth at will, allowing you to highlight it as a command when it appears as a command and as a non-command identifier when that is appropriate.

IX.C.1 Activating a base name, reserved as a `texcs` identifier, as a non-command identifier

For example, the `jdrlisting` package categorizes `\caption` as a built-in LaTeX `texcs` identifier and not as any other kind of identifier. As a result, when typeset by `jdrlisting`, `\caption` appears colored green.

However, `caption` is also the name of a key in the `listings` (and other) packages and, by default, when typeset by `jdrlisting`, `caption` will not appear properly formatted:

```
\begin{lstlisting}[float,caption=A floating example]
```
⟨✂56⟩

In order to correct the formatting of caption in code snippet ✂56, all you need to do is to declare caption as an emphasized identifier from third-party packages (other than Jim Ratliff's own packages).

One way to do this would be to add caption to the definition of \moreEmphNonJdrThirdParty[94] by editing the code of jdrlisting.sty, but this would ⓐ be a quasi-permanant change that could not be revised within the same document, ⓑ it would affect all documents that load jdrlisting, and ⓒ it would require the authority to edit jdrlisting.sty.

Instead, caption can be declared as non-JDR third-party emphasized identifier, for the purpose of the current document alone, by using the \addListEmphClassJdrLst command:[95]

```
\addListEmphClassJdrLst{caption}{1}
```
⟨✂57⟩

where the 1 in the second argument is the number of the class of emphasized identifiers reserved for identifiers from third-party packages.[96]

After executing \addListEmphClassJdrLst{caption}{1}, this identifier is now formatted as caption.

To revert, I use:

```
\deleteListEmphClassJdrLst{caption}{1}
```
⟨✂58⟩

After this reversion, the two identifiers are formatted as \caption and caption.

94 This is the command, from Table 7, that is associated with emphasized identifiers for third-party packages (other than Jim Ratliff's own packages).

95 See section VIII.A and Table 6.

96 See Table 5.

X Customizing the appearance of elements of listings whose appearance is controlled by jdrlisting

X.A Customizing colors

X.A.1 Customizing the color associated with each style and class of each type of identifier

Each style or style/class combination (when the style admits multiple classes) is associated with a named color with which the objects of that style are colored. See Table 8 for the name of the color ("Color to reassign") for each style or style/class combination as well as for the default value of that color.

To customize the color that is assigned to any style or style/class combination, just assign a different color to the "Color to reassign."

For example, any of the following are valid:

```
\definecolor{colorTeXCSStyleJdrLstB}{named}{teal}
\definecolor{colorCommentStyleJdrLst}{wave}{485}
\definecolor{colorTeXCSStyleJdrLstC}{rgb}{0.9,0.5,0.5}
```

⟨✂59⟩

Table 8: Name of color that can be customized, and its default value, for each style/class

Style	Class	Color to reassign	Default color
basicstyle		colorBasicStyleJdrLst	darkgray
identifierstyle		colorIdentifierStyleJdrLst	colorDarkSlateGrayJdrLst
texcsstyle	1	colorTeXCSStyleJdrLstA	colorForestGreenJdrLst
texcsstyle	2	colorTeXCSStyleJdrLstB	colorForestGreenJdrLst
texcsstyle	3	colorTeXCSStyleJdrLstC	magenta
emphstyle	1	colorEmphStyleJdrLstA	violet
emphstyle	2	colorEmphStyleJdrLstB	orange
keywordstyle		colorKeywordStyleJdrLst	violet
commentstyle		colorCommentStyleJdrLst	colorOrchidJdrLst

X.A.2 Customizing other colors

Two other colors that can be customized—in addition to those discussed in section X.A.1—are ⓐ the color of the line numbers that appear in the left margin of a lstlisting envi-

ronment and ⓑ the background color of a lstlisting environment. See Table 9.

Table 9: Names of other colors that can be customized, and their default values

Style/key	Color to reassign	Default color
numberstyle	colorNumberStyleJdrLst	colorHalfGrayJdrLst
backgroundcolor	colorBackgroundJdrLst	colorLightPinkJdrLst

X.B Customizing the character (or string) that prefixes the numeric identifier in the right margin of a jdrCodeSnip environment

In section III.E.4, I exhibited how a jdrCodeSnip environment produces a numeric label (e.g., ✂8) in the right margin. By default this number is prefixed by the character ✂.

This prefixing character (or string), viz., ✂, is defined in the jdrlisting package by the command \CodeSnipCharacterJdrLst.

You can change this character with a \renewcommand command, e.g.,

```
\renewcommand{\CodeSnipCharacterJdrLst}{SNIP}
```
⟨✂60⟩

which would produce references like "SNIP14".

X.C Customize the font sizes of the code and of the line numbers in the lstlisting environment

There are two font sizes associated with the lstlisting environment that you can change: ⓐ the size of the display-mode code itself[97] and ⓑ the size of the line numbers that are displayed along the left margin of lstlisting environment's frame.[98]

See Table 10 for the name of the command associated with each font size and its default value (as specified by the jdrlisting package).

To change either font size, use a \renewcommand command:

```
\renewcommand{fontSizeCommand}{fontSize}
```
⟨✂61⟩

[97] See section XI.B.

[98] See for example Listing 6.

Table 10: Font sizes in the lstlisting environment that you can change

Type of text	Command	Default value
Display-mode code	\lstFontSizeDisplay	\scriptsize
Line numbers	\lstFontSizeLineNumbers	\tiny

where

fontSizeCommand is either \lstFontSizeDisplay or \lstFontSizeLineNumbers

fontSize is a valid font size.

Valid font sizes are \tiny, \scriptsize, \footnotesize, \small, \normalsize, \large, \Large, \LARGE, \huge, and \Huge.[99]

For example, to make the displayed code one step larger, use:

```
\renewcommand{\lstFontSizeDisplay}{\footnotesize}
```
⟨✂62⟩

[[Note that I have inserted a \clearpage here solely to get this puppy to compile.]]

[99] See Reference guide at "Font sizes, families, and styles," Overleaf.

XI Discussion of selected implementation details of the jdrlisting package

XI.A Index of the commands and environments defined by the jdrlisting package

The following list provides references to either a discussion of, or a table of references to discussions of, the environments and commands defined by the jdrlisting package.

- \initializeLaTeXjdrLst: section III.B
- Table 11: Environments provided by the jdrlisting package and where they are discussed in this document
- Table 3: Commands provided by the jdrlisting package to emulate or extend listings syntax highlighting in text outside of \lstinline or lstlisting-type environments, and where they are discussed in this document
- Table 4: Commands provided by the jdrlisting package to emphasize/unemphasize the identifiers of particular JDR packages, and where they are discussed in this document
- Table 6: Commands provided by the jdrlisting package to add or delete a list of identifiers to a class or to move those identifiers between classes, and where they are discussed in this document
- Table 7: Commands provided by the jdrlisting package each of which stores a list of identifiers for a particular class of emph or texcs identifiers

Table 11: Environments provided by the jdrlisting package and where they are discussed in this document

Environment	§	Comment
jdrCodeSnip	III.E	Environment to display and number code snippets (or short segments) like equation
jdrLstListing	IV.C.2.2	Environment to typeset code that itself includes the lstlisting environment
jdrlstfloat	XI.C	(Deprecated) Environment to float a lstlisting listing[a]

[a] Use the technique in section III.F.2 instead.

XI.B Define basicstyle so that \lstinline text matches the surrounding text while making lstlisting listings have a given small text size

The listings-style basicstyle is the foundational style for all code displayed with the listings package in the sense that basicstyle is the "fallback style" for all other styles: each other style inherits the specifications of basicstyle except to the extent that that style overrides a basicstyle specification.[100]

A principle use case for the \lstinline command is to typeset a code snippet within an existing paragraph. Therefore the size of the font in which that snippet is typeset should match the font size of the surrounding text.

The lstlisting environment, on the other hand, typesets code to be set off from other text, i.e., in "display mode." The font size for lstlisting code thus is not constrained to match that of any other text. Indeed, it can be appropriate to choose a relatively small font size for this code so that an entire line of code (as naturally occurs in an editor) can be typeset on a single line of the document.

The jdrlisting package defines basicstyle to both:

- have \lstinline output match the font size of surrounding text, which it accomplishes by not defining a particular font size for in-line display;
- specify a particular font size, viz., \lstFontSizeDisplay, for display-mode code typeset by the lstlisting environment. By default, \lstFontSizeDisplay is set to \scriptsize.

These goals are accomplished through the following line of code:[101]

```
\lst@ifdisplaystyle\lstFontSizeDisplay\fi
```
⟨✂63⟩

See Listing 6 for the complete definition of basicstyle.

[100] See Table 12 and section A.C.1.

[101] "Package listings has two hooks TextStyle and DisplayStyle or a switch \lst@ifdisplaystyle, which can be used… to set a different font size in inline and displayed code listings." (See Heiko Oberdiek's answer to "Scaling inline code to the current font size," TEX Stack Exchange, February 20, 2014.)

Listing 6: The definition of basicstyle

```
\makeatletter
  \lstdefinestyle{styleBasicStyle}{%
    basicstyle=%
        \color{colorBasicStyleJdrLst}%
        \ttfamily
        \lst@ifdisplaystyle\lstFontSizeDisplay\fi% Assigns fontsize for display mode
 only }
\makeatother
```

XI.C Deprecated: The jdrlstfloat environment to float lstlisting listings like a figure or table

The jdrlstfloat environment is deprecated. Use the technique in section III.F.2 instead.

This package defines the jdrlstfloat environment so that you can "float" a code listing like a table or figure.[102]

The jdrlstfloat environment is defined using the \newfloat command from the float package.[103,104,105,106]

The \floatstyle for this environment is ruled, which causes the caption to be printed at the top of the float, with horizontal immediately above and below, and another horizontal rule at the bottom of the rule.[107]

The \floatname for this environment is "Listing". As a result, the caption of the seventh occurrence of this environment would begin "Listing 7".

---

102 "Among the features of LaTeX are 'floating' figures and tables that drift from where they appear in the input text to, say, the top of a page." Anselm Lingnau, "An Improved Environment for Floats," November 8, 2001, on page 1.

103 This was the suggestion in Arun Debray's answer to "Float for lstlisting," TeX Stack Exchange, November 20, 2015.

104 The float package "[i]mproves the interface for defining floating objects such as figures and tables. Introduces the boxed float, the ruled float and the plaintop float. You can define your own floats and improve the behaviour of the old ones. The package also provides the H float modifier option of the obsolete here package." Its documentation is: Anselm Lingnau, "An Improved Environment for Floats," November 8, 2001.

105 The float's definition includes \def\jdrlstfloatautorefname{Listing}, which I understand, from Arun Debray's answer to "Float for lstlisting," TeX Stack Exchange, November 20, 2015, is "needed for hyperref/autoref,'' (though I haven't looked into this in order to understand this claim).

106 If you use the cleveref package, you'll probably want to add the following in your preamble: \crefalias{jdrlstfloat}{Listing}, per Lucas Werkmeister's comment to Arun Debray's answer to "Float for lstlisting," TeX Stack Exchange, November 20, 2015, that: "for cleveref support, add \crefalias{lstfloat}{listing}."

107 See Anselm Lingnau, "An Improved Environment for Floats," November 8, 2001, on page 2.

The jdrlstfloat environment does not by itself trigger any command or environment from the listings package. Rather it provides a wrapper inside of which such a command or environment can be inserted.[108]

XI.C.1 Syntax for the jdrlstfloat environment

A typical invocation of the jdrlstfloat environment includes (after the obligatory \initializeLaTeXjdrLst of course):

- invoking the jdrlisting environment;
- invoking the lstlisting environment;
- inserting the desired LaTeX code to typeset;
- inserting a \caption{*someCaption*} command;
- inserting a \label{lsting:*someLabel*} command.[109]

This sequence is shown in the code below:

Listing 7: A typical invocation of jdrlstfloat (now deprecated).

```
\documentclass{article}
\usepackage{jdrlisting}
\begin{document}
\initializeLaTeXjdrLst
\begin{jdrlstfloat}
\begin{lstlisting}
INSERT LaTeX CODE HERE, e.g., \newcommand
\end{lstlisting}
\caption{This is the \LaTeX{} code that produces something of interest}
\label{lsting:labelForThisListing}
\end{jdrlstfloat}
\end{document}
```

The output of which appears as Listing 8:

Listing 8: This is the LaTeX code that produces something of interest

```
INSERT LaTeX CODE HERE, e.g., \newcommand
```

108 This is similar to the table environment, which can be a wrapper inside of which a tabular environment is inserted.

109 There is nothing magic or mandatory about my suggestion of using lsting as the beginning of the marker phrase. It is meant to be analogous to the common practice of, for example, using fig: as the beginning of a label for a figure or using tab: for the beginning of a label for a table. ("Since you can use exactly the same commands to reference almost anything, you might get a bit confused after you have introduced a lot of references. It is common practice among LaTeX users to add a few letters to the label to describe *what* you are referencing." LaTeX/Labels and Cross-referencing, Wikibooks.)

XI.C.2 Producing a "List of Listings"

To produce a "List of Listings" akin to "List of Figures" or "List of Tables,"[110] include the following command:[111]

```
\listof{jdrlstfloat}{List of Listings}
```
⟨✂<64⟩

XI.C.3 Qualifications regarding and alternatives to the jdrlstfloat environment

When using the jdrlstfloat environment, the "Listings" are numbered continuously from the beginning to the end of the document; the number is never reset at any sectional unit, such as a new chapter. This is particularly appropriate for an article-class document.[112]

The jdrlstfloat environment is certainly not necessary to create a floating listing. The listings package also allows a float key to the lstlisting environment. You can also specify a caption key and a label key directly to lstlisting.[113] I have done nothing to investigate the pro and cons of using these capabilities built-in to listings vis-à-vis my jdrlstfloat methodology.[114]

---

110 For more on producing a "List of Figures" and/or "List of Tables" see "Lists of tables and figures," Overleaf.

111 Typically this command would immediately, or almost immediately, follow the \tableofcontents command, perhaps with \listoffigures and/or \listoftables intervening between ⓐ \tableofcontents and ⓑ \listof{jdrlstfloat}{List of Listings}.

112 You might find this undesirable if, for example, you're using the report or book document style, both of which reset the counters for Figures and Tables at a new chapter. The float package *does* provide for such counter resets; it is this jdrlisting package that does not. (I could introduce an option to this package, but I currently use only the article class so doing so is not a high priority.) You could replicate the definition of jdrlstfloat, and rename it, and add the optional parameter [chapter], as in \newfloat{newJdrLstFloat}{htbp}{lop}[chapter]. See the discussion of the optional *within* parameter in Anselm Lingnau, "An Improved Environment for Floats," November 8, 2001, on page 2.

113 See ListingsDocs, §§ 4.3 and 4.9.

114 However, note that Radoslav reported a problem in "Float for lstlisting," TEX Stack Exchange, November 20, 2015: "Float attribute… is OK only for lstlisting [if] count of lines is smaller than half page.… If count of lines is greater than half page, lstlisting is alone on the page."

A Selective summary of features and usage of the listings package

The listings package, maintained by Jobst Hoffman, is:[115]

> a source code printer for LaTeX. You can typeset stand alone files as well as listings with an environment similar to verbatim as well as you can print code snippets using a command similar to \verb.

A.A The listings package can present code either in-line or in display mode

The listings package distinguishes broadly between two modes of presentation of highlighted code: ⓐ code snippets and ⓑ code segments:[116,117]

- A code snippet is placed inside a paragraph, i.e., rendered "in line." It is typeset with the command \lstinline.[118]
- A code segment appears as one or more separate paragraphs, i.e., as "displayed code." It is displayed in the lstlisting environment.[119]

A.B The types of code strings between which the listings package distinguishes

Within a particular set of code, whether to be presented as a snippet or a segment, the listings package distinguishes between several types of code entities.

A.B.1 Strings of code distinguished by being specially delimited: comments and strings

First, listings identifies strings of code that are distinguished by virtue of how they are delimited. There are two types of such code strings:

---

115 Carsten Heinz, Brooks Moses, and Jobst Hoffmann, "The Listings Package," September 2, 2018, version 1.7, tinyurl.com/listingsDocs, hereafter "ListingsDocs."

116 ListingsDocs, § 1.2 on page 4. "Three types of source codes are supported: code snippets, code segments, and listings of stand alone files. Snippets are placed inside paragraphs and the others are separate paragraphs—the difference is the same as between text style and display style formulas."

117 It also recognizes listings of standalone files, but I ignore that part of its capabilities.

118 See section A.D.3 for discussion of \lstinline.

119 See section A.E.1 for discussion of the lstlisting environment.

- Logically first are comments. They must be identified first; otherwise the contents of a comment could be erroneously categorized as another type of code. Comments are formatted with the `commentstyle` style.[120]
- Second are strings in the sense in which the language/dialect dialect defines a string.[121] Strings are formatted with the `stringstyle` style.[122] (Strings in the `listings` sense, viz., as delimited strings, are not present in LaTeX; I will largely/entirely ignore them henceforth.)

A.B.2 Remaining one-word code elements after comments and strings are otherwise accounted for

After removing all instances of strings of code that are so delimited (i.e., comments and strings), we look at what's left, disaggregated into individual "words."[123]

Within the set of individual words, a subset is what `listings` calls "identifiers," which must begin with a letter and be followed by alpha-numeric characters.[124] Any individual word not qualifying as an identifier (e.g., `+` or `1776`) is in the "everything else" category and is formatted according to `basicstyle`.[125] Note: in some cases it's a little more complicated than this.[126]

The identifiers can be partitioned (in the particular sense of no word should be assigned

---

120 ListingsDocs, § 4.6 on page 30. See section A.C.3.6 and Table 12.

121 I acknowledge the possible ambiguity in the use of "string'' to mean both ⓐ a string of code and ⓑ a string of code that also is a string in the programming language's sense.

122 ListingsDocs, § 4.6 on page 30. See also Table 12.

123 By "word," I simply mean sequences of characters with no internal white spaces; I do not imply that they contain even one letter. ("[W]hite space characters are prohibited inside keywords.'' ListingsDocs, § 3.2 on page 23.) See also "How to emphasize within a listing two successive identifiers separated by a space?," TeX Stack Exchange, September 6, 2011.

124 "All identifiers (keywords, directives, and such) consist of a letter followed by alpha-numeric characters (letters and digits).'' Which characters are considered letters, digits, and "other" is specified by the documentation's Table 2 and, as well, by the keys `alsoletter`, `alsodigit`, and `alsoother`. (ListingsDocs, § 4.18 on page 45.) See also jubobs' answer to "How can I get identifier style to apply to '%' (in a Perl listing)?," TeX Stack Exchange, November 6, 2013.

125 See section A.C.3.2.

126 For example, although the "word" `1776isaplace` is not an identifier—for the sufficient reason that it does not start with a letter—neither is that 12-character string a nonidentifier. Instead, `1776` is formatted as a nonidentifier (i.e., according to `basicstyle`) while the remaining letters are formatted as an identifier (i.e., according to `identifierstyle`), notwithstanding that the digits part and the letters part are not separated by a space or any other delimiter. See section A.F.3.

to more than one category)[127]—by a combination of ⓐ the definition of the language and dialect[128] and ⓑ declarations by the author—into the following four categories:[129]

A.B.2.1 TEX control sequences

TEXcontrol sequences are defined—only when TEX is the specified language—by a set of identifiers (and therefore necessarily do not include the leading backslash) that have been declared by the language/dialect or later uses of the `texcs` and/or `moretexcs` keys.[130]

Although defined by identifiers, a string of code is not considered a TEX control sequence unless it begins with a backslash and is immediately followed by one of these identifiers.[131]

Note a conflict if the same term is defined both as a TEX control sequence and as an emphasized identifier:[132] If a term `foo` exists as both ⓐ a TEX control sequence via `moretexcs` and ⓑ an emphasized identifier via `emph`, the control sequence `\foo` will be formatted as `\emphstyle`.[133] Managing these conflicts is discussed in section IX.C.

TEX control sequences are formatted with the `texcsstyle` style.[134]

---

127 A partition of a set is a grouping of the set's elements into non-empty subsets, in such a way that every element is included in one and only one of the subsets. ("Partition of a set," Wikipedia.)

128 See Carsten Heinz et al. "Language, Style and Format drivers for `listings`," September 2, 2018, version 1.7, in which the relevant characteristics of each language and dialect are defined.

129 These identifiers can and should be partitioned, but this is not forced, only strongly recommended: "One final hint: Keep the list of identifiers disjoint. Never use a keyword in an 'emphasize' list or one name in two different lists. Even if your source code is highlighted as expected, there is no guarantee that it is still the case if you change the order of your listings or if you use the next release of this package.'' (ListingsDocs, § 2.8 on page 20.)

130 See section A.C.4.1 and Table 13.

131 When typesetting (LA)TEX code, it is better to add additional control sequences, i.e., which start with a \, as a TEXcontrol sequence rather than as a keyword. As a TEXcontrol sequence, the identified word will be highlighted only when it is preceded by a \. If that word were instead a keyword, it would be highlighted as a control sequence even when it is encountered without being preceded by a \.

132 I have not tested whether the same conflict arises if you define a term to be both a TEX control sequence and as a keyword.

133 "Bug: `texcs`…interferes with other keyword lists. If, for example, `emph` contains the word `foo`, then the control sequence `\foo` will show up in `emphstyle`." (ListingsDocs, § 4.6 on page 31.)

134 If the `texcsstyle` is not specified, the formatting defaults to `keywordstyle`. See also section A.C.3.4 and Table 12.

A.B.2.2 Keywords

Keywords are defined by a set of identifiers that have been declared by the language/dialect or later uses of the keywords and/or morekeywords keys.[135] Keywords are formatted with the keywordstyle style.[136,137]

A.B.2.3 Emphasized identifiers

Emphasized identifiers are defined by a set of identifiers that have been declared by the language/dialect or later uses of the emph and/or moreemph keys.[138] Emphasized identifiers are formatted with the emphstyle.[139]

A.B.2.4 Non-emphasized identifiers

Non-emphasized identifiers is the catch-all category into which any remaining identifiers are passively assigned by omission. These all-else words are formatted with the identifierstyle.[140]

See Table 12 for a summary of the style that formats each of the above four categories of one-word code elements.

The first three of these four categories of one-word code elements—viz., TEXcontrol sequences, keywords, and emphasized identifiers—can each be further refined for formatting purposes by optionally dividing its one-word elements into separate classes.[141] When a category has more than one class, both ⓐ the key to assign a style to the category and

---

135 See section A.C.4.2 and Table 13.

136 If the keywordstyle is not specified, the formatting defaults to a bolded version of basicstyle, i.e., \bfseries layered on toap of basicstyle. See also Table 12.

137 ListingsDocs never defines the criteria for what should or should not be a keyword, other than to suggest that keywords are perhaps more fundamental than function names: "[f]or many programming languages it is sufficient to specify keywords and standard function names, comments, and strings," compounded by "[t]here isn't must to say about keywords. They are defined like identifiers you want to emphasize." (ListingsDocs, § 3.2, on page 22.)

138 See section A.C.4.3 and Table 13.

139 If the emphstyle is not specified, the formatting defaults to basicstyle. See also Table 12.

140 If the identifierstyle is not specified, the formatting defaults to basicstyle. See also Table 12.

141 See ListingsDocs, § 4.6 on pages 30–31.

ⓑ the key to add words to the list of words in the category use an optional argument [*n*] to identify the particular class at which the *key*=*value* is aimed.[142]

Table 12: The listings style-name key that governs the format of each category, and class, of code element

Category[a]	Controlling style	Fallback style #1	Fallback style #2
Comments	commentstyle	\itshape·basicstyle[b]	
TEX control sequences	texcsstyle	keywordstyle	basicstyle
Keywords	keywordstyle	\bfseries·basicstyle[c]	
Emphasized identifiers	emphstyle	basicstyle	
Nonemphasized identifiers	identifierstyle	basicstyle	
Nonidentifiers	basicstyle		

[a] The string category is omitted from this table because it's not used with LATEX and as a result I can't easily test any claims about how it would be formatted.

[b] If commentstyle is not defined, the formatting falls back to basicstyle enhanced by \itshape.

[c] If keywordstyle is not defined, the formatting falls back to basicstyle enhanced by \bfseries.

A.C A selective summary of key-value pairs recognized by the listings package

Several commands and the an environment accept input in the form of a list of comma-separated pairs of the form *key*=*value*. These list-accepting commands and environment include:

- \lstset (see section A.D.1)
- \lstdefinestyle (see section A.D.2)
- \lstinline (see section A.D.3)
- the lstlisting environment (see section A.E.1)

A.C.1 Commands that assign values to keys are additive to previous such commands

[[I'm still awaiting authority for this. In the meantime note: ⓐ Defining a style affects the values of only the keys explicitly named in the style definition. The values of other keys

[142] See sections A.C.3.3 and A.C.4.2 (keywords); sections A.C.3.4 and A.C.4.1 (TEX control sequences); and sections A.C.3.5 and A.C.4.3 (emphasized identifiers).

are left untouched.[143] ⓑ Keys whose values are set via \lstset generally keep their values up to the end of the current environment or group. Afterwards the previous values are restored.[144]]]

A.C.2 Declaring the language and, optionally, dialect: language=[*dialect*] *language*

The subject programming language and optionally dialect are specified with a key/value pair:

language=[*dialect*] *language* ⟨✂65⟩

In particular, for LaTeX this takes the form:

language=[LaTeX] TeX ⟨✂66⟩

You must put braces around the value if a value with optional argument is used inside an optional argument; e.g.,

```
\lstinline[language={[LaTeX]TeX}]|\newcommand{\myMacro}{myString}|
```
⟨✂67⟩

A.C.3 Keys that take formatting commands for a *style* value

A.C.3.1 Some keys have *style* values

Many keys, such as basicstyle, keywordstyle, texcsstyle, emphstyle, commentstyle, and identifierstyle are associated with values of type *style*.[145] (Terminological warning:

[143] "Keys not used in such a definition are untouched by the corresponding style selection.'' (ListingsDocs, § 3.1 on page 22.)

[144] There are exceptions to the general rule regarding some optional parameters: "All parameters set via \lstset keep their values up to the end of the current environment or group. Afterwards the previous values are restored. The optional parameters of the two pretty-printing commands and the lstlisting environment take effect on the particular listing only, i.e. values are restored immediately. For example, you can select a main language and change it for special listings." (ListingsDocs, § 2.3 on page 12.)

[145] See sections A.C.3.2, A.C.3.3, A.C.3.4, A.C.3.5, and A.C.3.6, and A.C.3.7, respectively.

the documentation uses "style" with two distinct meanings.[146])

Though the documentation never states explicitly and generally what values *style* can take, my inductive conclusion is that *style* can take any command that could be used to format text, or string of concatenated such commands, with the understanding that, at least for the most part, these commands would be *switches* rather than commands that take an argument.[147]

Examples of what can be substituted for *style* include:

```
\footnotesize%  Font size
\small%         Font size
\itshape%       Switch that turns on italics through remainder of group
\bfseries%      Switch that turns on bold formatting
\ttfamily%      Switch that turns on fixed-wdith/monotype font
\color{black}\bfseries\underbar% An example of concatenation of commands
```

⟨✂68⟩

A.C.3.2 basicstyle=*style*

The syntax for the basicstyle key is:

basicstyle=*style* ⟨✂69⟩

where *style* is a command that can be used to format text, or a string of concatenated such commands, as described further in section A.C.3.1.[148]

As I explain in section A.B.2, listings acknowledges a set of one-word entities (like + or 1776) that are not identifiers. These nonidentifiers are formatted according to

[146] Note that the documentation uses "style" in two distinct ways: ⓐ First, style (as in section A.C.3.1) is a particular kind of value in a *key*=*value* pair, where *key* is one of a set of particular keys that include "style" in the name of the key (e.g., basicstyle, keywordstyle, texcsstyle, emphstyle, commentstyle , and identifierstyle). In this sense, *style* is a string of commands related to formatting in the narrow sense of specifying a font family, font weight, font size, or color. ⓑ Second, "style" refers to a named style (as in style=*stylename*), which is a collection of *key*=*value* pairs—a totally different animal than the first use of "style." Moreover, such a *key*=*value* pair need not have be formatting related in this narrow sense at all. See, e.g., section A.C.5.

[147] "In general, the *very last* command may read exactly one argument, namely some material the package typesets. There's one exception. The last command of basicstyle must not read any tokens—or you will get deep in trouble." (ListingsDocs at page 6. Emphasis in original.)

[148] In particular, note: "In general, the *very last* command may read exactly one argument, namely some material the package typesets. There's one exception. The last command of basicstyle must not read any tokens—or you will get deep in trouble." (ListingsDocs at page 6. Emphasis in original.)

basicstyle.[149]

Moreover, as Table 12 makes clear, basicstyle is also the ultimate fall-back style for every other category of code element.

Importantly, the effect of the basicstyle specification on the formatting of a particular non-nonidentifier category of code element is not limited to the extreme case in which the controlling style for that code element type (e.g., keywordstyle for keywords) is not specified. Even when a code-category-specific style is specified, that style may not specify a value for every dimension of the style.[150] In that case, the value of a dimension not specified by the controlling style would be determined by a fall-back style if the fall-back style did specify a value for that dimension.

For example, if basicstyle=\ttfamily, and if keywordstyle does not specify a font family, then the specification of \ttfamily from basicstyle will control for keywords even if the keywordstyle is defined for other dimensions. Consider the following preamble:

```
\usepackage{listings}
\usepackage{xcolor}
\lstset{%
        basicstyle=\color{blue}\ttfamily,%
        keywordstyle=\color{red},%
        morekeywords={someKeyword}%
        }
```

⟨✂70⟩

Any occurrence of someKeyword within a lstlisting environment or a \lstinline command will be typeset as both ⓐ colored red, as defined by keywordstyle, and ⓑ monospaced, defined by the specification by basicstyle of \ttfamily. In other words, because keywordstyle was defined but did not touch the font-family dimension, that dimension was determined by the fall-back style basicstyle.

The basicstyle key defaults to {}.[151]

149 See also Table 12.

150 There are multiple, orthogonal dimensions of a style, such as font family, font weight, font shape, and font color.

151 ListingsDocs, § 4.6 on page 30.

A.C.3.3 keywordstyle=*style*

The keywordstyle is the style for keywords, which are assigned with the morekeywords key. keywordstyle is also the fall-back style for TEX control sequences.[152]

The syntax for the keywordstyle key is:

keywordstyle=[*n*]*style* ⟨✂71⟩

where

- *n* is an optionally specified natural number that identifies the number of the class of keywords to which the style should be applied; and
- *style* is a command that can be used to format text, or a string of concatenated such commands, as described further in section A.C.3.1.

The keywordstyle defaults to \bfseries, which is the switch for **bold** output, layered on top of the fall-back style basicstyle.

A.C.3.4 texcsstyle=*style* (TEX specific)

The texcsstyle key determines the style of TEX control sequences declared via the moretexcs key. The syntax of the texcsstyle is:

texcsstyle=[*]⟨*n*⟩*style* ⟨✂72⟩

where:

- the optional * highlights the backslash in front of the control sequence name;[153]
- *n* is an optionally specified natural number that identifies the number of the class of TEX control sequences to which the style should be applied; and
- *style* is a command that can be used to format text, or a string of concatenated such commands, as described further in section A.C.3.1.

---

[152] See section A.C.3.4 and Table 12.

[153] "Note that this option is set for all texcs lists." (ListingsDocs, § 4.6, at page 31.) (I'm not sure what this means. Does it mean the * is superfluous?)

If texcsstyle is not specified, the formatting of TEX control sequences falls back to keywordstyle.

A.C.3.5 emphstyle=*style*

The emphstyle key determines the style of emphasized identifiers declared via the emph key. The syntax of the emphstyle key is:

emphstyle=[*n*]*style* ⟨✂73⟩

where

- *n* is an optionally specified natural number that identifies the number of the class of emphasized identifiers to which the style should be applied; and
- *style* is a command that can be used to format text, or a string of concatenated such commands, as described further in section A.C.3.1.

If emphstyle is not specified, the formatting of emphasized identifiers falls back to basicstyle.[154]

A.C.3.6 commentstyle=*style*

Comments, as defined by the language definition, are formatted according to the commentstyle style, which defaults to \itshape, which is the switch for *italics*, layered on top of basicstyle.

The syntax of the commentstyle key is:

commentstyle=*style* ⟨✂74⟩

where *style* is a command that can be used to format text, or a string of concatenated such commands, as described further in section A.C.3.1.

[154] See Table 12.

A.C.3.7 identifierstyle=*style*

The syntax of the identifierstyle key is:

identifierstyle=*style* ⟨✂<75⟩

where *style* is a command that can be used to format text, or a string of concatenated such commands, as described further in section A.C.3.1.

The value of identifierstyle key, if present, controls the formatting of identifiers that do not otherwise have alternative formatting defined—i.e., identifiers that are not a TEX control sequence, keyword, or emphasized identifier.[155]

If identifierstyle is not defined, the formatting of these otherwise-unspecified identifiers is controlled by basicstyle.[156]

A.C.4 Keys that assign identifiers to categories of identifiers: keywords, TEX control sequences, and emphasized identifiers

The listings package is designed to perform syntax highlighting on code, but that of course requires listings to understand enough about the code to know which parts to highlight differently than other parts.

I explained in section A.B that the listings package distinguishes first parts of code that, by virtue of their delimiters, are classified as either comments or as strings.[157] The specification of the delimiters that allow listings to make this classification are made in the definition of the language and dialect. Because I'm not interested in defining a language/dialect from scratch, I take these specifications as given.

Following the classification of code strings as ⓐ comments or ⓑ programmatic strings, listings then concerns itself with the one-word pieces that are left.[158] In order to differentially format those words, it counts on a combination of ⓐ the definition of the language/dialect and ⓑ other specifications (e.g., by additional loaded packages or by spec-

[155] See section A.B.2.

[156] See Table 12.

[157] See section A.B.1.

[158] See section A.B.2.

ifications made by the document author) to put those remaining words into categories.[159] the listings package then applies specified formatting to each category of words.[160]

These specifications of which words belong to which of the three main categories—keywords, TEX control sequences, and emphasized identifiers—ultimatly come, for each category, from a trio of related keys: ① a key to define the set of words that belong to the category, ② a key to supplement that list, and ② a key to delete that list. That said, here I will only concern myself with the key that supplements an existing list.[161]

As well, each of these three categories can be further refined into classes. Each class of a category can be formatted separately from every other class of that category.

A.C.4.1 moretexcs to define or supplement the list of TEX control sequences

The syntax for the moretexcs key is:[162]

moretexcs=[n]{cs_1,...,cs_n} ⟨✂76⟩

where:

- n is an optionally specified natural number that identifies the number of the class of TEX control sequences to which the identifier should be added; and
- the cs_i are the TEX control sequence (*without* a prefixing backslash) that are to be added to the list for that class of TEX control sequences.

159 It's not clear, but I believe that these assignments are specific to a language/dialect. Hence, I suppose you need to declare the language/dialect before assigning values to these keys. But see, regarding a key keywordsprefix: "[i]f used 'standalone' outside a language definition, the key might work only after selecting a nonempty language (and switching back to the empty language if necessary)." (ListingsDocs, § 4.18 on page 44.)

160 The formatting specified for each category is provided by the keys discussed in section A.C.3.

161 Indeed, the documentation warns against use of the keywords key: "The use of keywords is discouraged since it deletes all previously defined keywords in the list and is thus incompatible with the alsolanguage key.'' (ListingsDocs, § 4.18 on page 44.)

162 ListingsDocs, § 4.18 on page 44.

Table 13: The keys that control formatting and word assignment to each category of code

Category[a]	Defines style for category	Assigns words to category
TEX control sequences	`texcsstyle=[*][n]`*style*	`moretexcs=[n]{`$cs_1,\ldots,cs_n$`}`
Keywords	`keywordstyle=[n]`*style*	`morekeywords=[n]{`$w_1,\ldots,w_n$`}`
Emphasized identifiers	`emphstyle=`*style*	`moreemph=[n]{`$w_1,\ldots,w_n$`}`
Comments	`commentstyle=`*style*	—
Everything else	`identifierstyle=`*style*	—

Notes

n:	an optionally specified natural number that identifies the class number to which the style should be applied.
cs_i	a control sequence—*without* the backslash.
w_i	a word to assign to the category list.

[a] The string category is omitted from this table because it's not used with LATEX and as a result I can't easily test any claims about how it would be formatted.

A.C.4.2 `morekeywords` to define or supplement the list of keywords

The syntax for the `morekeywords` key is:[163]

`morekeywords=[`*n*`]{`$w_1,\ldots,w_n$`}` ⟨✂77⟩

where

- *n* is an optionally specified natural number that identifies the number of the class of keywords to which the identifier should be added; and
- the w_i are the identifiers to assign to the list of keywords.

[163] ListingsDocs, § 4.18 on page 44.

A.C.4.3 moreemph to define or supplement the list of identifiers to emphasize

The syntax for the moreemph key is:[164]

moreemph=[n]{$w_1, \ldots, w_n$} ⟨✂78⟩

where

- n is an optionally specified natural number that identifies the number of the class of emphasized identifiers to which the identifier should be added; and
- the w_i are the identifiers to assign to the list of keywords.

A.C.4.4 A control sequence whose base name is either a keyword or an emphasized identifier cannot be formatted as TEX control sequence

It can happen that the same identifier, e.g., foo can be ⓐ the base name of a TEX control sequence and ⓑ the name of some kind of parameter (such as a key). Each of these usages deserves separate formatting. Unfortunately, declaring foo to be either a keyword or an emphasized identifier prevents it from also being formatted as a control sequence (when it's preceded by a backslash and declared a texcs).

This conflict, and how to manage it, is discussed in section IX.C.

A.C.5 A set of *key*=*value* pairs can be assigned to a named "style," style=*stylename*, for later retrieval and implementation

In a second type of meaning of "style,"[165] a style refers to a set of *key*=*value* pairs, which set can thereby be easily referenced and implemented.

The style key refers to a *stylename*, which is associated with a set of key-value pairs. The *stylename* becomes associated with a set of *key*=*value* pairs by passing a comma-separated list of *key*=*value* pairs to the \lstdefinestyle command.[166]

A named style can be invoked, for example, by \lstset{style=someNamedStyle}, where someNamedStyle is a previously named style.

---

164 ListingsDocs, § 4.6 on page 31.

165 See footnote 146 for a discussion of the ambiguity of "style" within ListingsDocs.

166 See section A.D.2.

Defining a style affects the values of only the keys explicitly named in the style definition. The values of other keys are left untouched.[167]

A.D Commands defined in the listings package

A.D.1 \lstset

The syntax for \lstset is:

\lstset{*key=value list*} ⟨✂79⟩

\lstset sets the values of the specified keys (and only the specified keys, leaving other keys untouched) until the end of the current environment or group. (If you want to change settings for a single listing, use an optional argument of lstlisting or \lstinline.[168])

Keys whose values are set via \lstset generally keep their values up to the end of the current environment or group. Afterwards the previous values are restored.[169]

A.D.2 \lstdefinestyle

To define a style:

\lstdefinestyle{*style name*}{*key=value list*} ⟨✂80⟩

Defining a style affects the values of only the keys explicitly named in the style definition. The values of other keys are left untouched.[170]

A.D.3 \lstinline

The syntax of \lstinline is:

\lstinline[*key=value list*]*<char><verbatim code><char>* ⟨✂81⟩

---

[167] "Keys not used in such a definition are untouched by the corresponding style selection.'' (ListingsDocs, § 3.1 on page 22.)

[168] ListingsDocs, § 2.5 on page 14.

[169] There are exceptions to the general rule regarding some optional parameters: "All parameters set via \lstset keep their values up to the end of the current environment or group. Afterwards the previous values are restored. The optional parameters of the two pretty-printing commands and the lstlisting environment take effect on the particular listing only, i.e. values are restored immediately. For example, you can select a main language and change it for special listings." (ListingsDocs, § 2.3 on page 12.)

[170] "Keys not used in such a definition are untouched by the corresponding style selection.'' (ListingsDocs, § 3.1 on page 22.)

where <*char*> refers to a single character that is not found in <*verbatim code*>.[171] This single character both immediately precedes and immediately follows the string of code to be rendered verbatim.[172]

A.D.4 \lstlistoflistings

\lstlistoflistings command prints the list of listings.[173]

A.D.5 \lstnewenvironment

The \lstnewenvironment command is used to "define your own pretty-printing environments."[174] Its syntax comes from LaTeX's \newenvironment command:[175,176]

```
\lstnewenvironment{name}[n_args][v_default]%
    {starting code}%                                    ⟨✂82⟩
    {ending code}
```

[171] It's better to avoid using an opening square bracket (i.e., [) as the delimiter because \lstinline has to scan for that character because it would delimit the command's optional argument. See ListingsDocs at § 4.2. ("Since the command first looks ahead for an optional argument, you must provide at least an empty one if you want to use [as <*character*>.")

[172] This convention, which leaves the delimiter undesignated ex ante, allows the delimiter to be chosen according to the string desired to be rendered verbatim.

[173] It's possible that a listing gets an entry in the list of listings only if the caption is defined in the \begin{lstlisting} command (as opposed to defined in a floating environment). See ListingsDocs on page 18 under section "Captions."

[174] ListingsDocs, § 4.16.

[175] ListingsDocs, § 4.16.

[176] For the syntax for LaTeX's \newenvironment command, see, for example, "Defining a new environment" in "Environments," Overleaf.

where

name is the name of the environment to be defined;[177]

n_{args} is the number of arguments; if there are no arguments, omit [n_{args}];

$v_{default}$ is the default value, if any, for the optional argument, if any. If you do not want an optional argument, omit the [$v_{default}$]. If you want an optional argument, but don't want to specify a default value for it, just use [];

starting code The text substituted for every occurrence of \begin{*name*}. A parameter of the form #*n* will be replaced by the text of the *n*-th argument when this substitution takes place;

ending code The text substituted for every occurrence of \end{*name*}. It may not contain any argument parameters.[178]

Although the documentation does not say much more than what this syntax is, it appears that the new environment in effect wraps an instance of the lstlisting environment.

For an application of using \lstnewenvironment, see section IV.C.2.2.

A.E Environments defined in the listings package

A.E.1 The lstlisting environment

The lstlisting environment typesets the code in between the \begin{lstlisting} and \end{lstlisting} commands as a displayed listing. Its syntax is:

```
\begin{lstlisting}[key=value list]
%   Some code                                    ⟨✂83⟩
\end{lstlisting}
```

You can assign a caption and/or label to a lstlisting environment even if it not in a float:

```
\begin{lstlisting}[caption={Useless code},label=useless]    ⟨✂84⟩
```

A.F Other topics related to the listings package

A.F.1 Certain characters sometimes need to be escaped

This discussion has been moved above to section IV.A.

A.F.2 The style remembers the *name* of a color, not the actual color at the time the style is defined

Suppose you define a style and, further, as part of that definition you specify the name of a color. What the style will remember is the *name* of that color, not the actual color assigned to that name at the time of the definition.

```
\newcommand{}{}
```

Changing the color to red.

```
\newcommand{}{}
```

```
\documentclass{article}
\usepackage{listings}
\usepackage{xcolor}
\definecolor{texcsColor}{named}{blue}
\lstset{language=[LaTeX]TeX,texcsstyle=\color{
    texcsColor}\ttfamily}
\begin{document}
\begin{lstlisting}
\newcommand{}{}
\end{lstlisting}
Changing the color to \textcolor{red}{red}.
\definecolor{texcsColor}{named}{red}
\begin{lstlisting}
\newcommand{}{}
\end{lstlisting}
\end{document}
```

A.F.3 In a string like "1776isayear," the numeric part is formatted according to basicstyle while the alphabetic part is formatted according to identifierstyle

In section A.B.2, I said that:

> After removing all instances of strings of code that are so delimited (i.e., comments and strings), we look at what's left, disaggregated into individual "words." Within the set of individual words, a subset is what listings calls "identifiers," which must begin with a letter and be followed by alpha-numeric characters. Any individual word not qualifying as an identifier (e.g., + or 1776) is in the "everything else" category and is formatted according to basicstyle.

That picture is too simple in the case of a word that begins with digits but ends with a sequence of letters. For example, although the "word" 1776isaplace is not an identifier—for the sufficient reason that it does not start with a letter—neither is that 12-character string a nonidentifier. Instead, 1776 is formatted as a nonidentifier (i.e., according to basicstyle) while the remaining letters are formatted as an identifier (i.e., according to

identifierstyle), notwithstanding that the digits part and the letters part are not separated by a space or any other delimiter. This behavior is illustrated in the below code.

Note as well that a string like Two345 would seem to satisfy the intent of the following description of identifiers: "All identifiers…consist of a letter followed by alpha-numeric characters (letters and digits)."[179] Yet, in the example directly below, the letters are formatted according to identifierstyle while the digits are formatted according to basicstyle.[180]

```
This is a place I know.
2 + 2 = 1776isaplace
Two+2=Four
Two2=Four
Two345
```

```
\documentclass{article}
\usepackage{listings}
\usepackage{xcolor}
\lstset{basicstyle=\color{blue}\ttfamily,%
        identifierstyle=\color{red}%
}
\begin{document}
\begin{lstlisting}
This is a place I know.
2 + 2 = 1776isaplace
Two+2=Four
Two2=Four
Two345
\end{lstlisting}
\end{document}
```

179 ListingsDocs, § 4.18 on page 45.

180 Perhaps this reflects not ① a problem with the stated characterization of what is and is not an identifier but rather ② a divergence between ⓐ identifiers and ⓑ what characters are formatted according to identifierstyle.