

MANUAL for

THE *jlcode* PACKAGE Version 5.1

November 26, 2020

(Version 1.0: January 25, 2018)

<https://github.com/wg030/jlcode>

Copyright 2018–2020 GitHub user wg030

Contents

1	Introduction	2
2	License	2
3	Package Options	3
3.1	Current Package Options	3
3.2	Obsolete Package Options	4
4	How to Insert Code in Your Document	5
4.1	Inserting Code with <code>autoload=true</code>	5
4.2	Inserting Code with <code>autoload=false</code>	6
5	How to Increase the Font Size	7
6	Themes	8
6.1	Example with <code>theme=default</code>	9
6.2	Example with <code>theme=grayscale</code>	10
6.3	Example with <code>theme=default-plain</code>	11
6.4	Example with <code>theme=grayscale-plain</code>	12
6.5	Example with <code>theme=darkbeamer</code>	13
7	Known Managable Issues	14

1 Introduction

The `jrcode` package (*jrcode.sty*) provides a language definition for the programming language Julia for the listings package as well as five different style definitions. Loading this package is the easiest way to display Julia code within your document when you want to make use of the listings interface.

This package takes especially care of correctly displaying code that contains the most common unicode characters such as greek letters, superscripts or mathematical symbols, which Julia allows as valid characters for identifier names. Moreover all keywords, literals, built-ins, macros, functions and string types that belong to Julia's standard library were generated by a script (*createlists.jl*) and then included in the language definition of this package.

Alongside the actual language definition a few nice looking styles were defined, too, which will help you to highlight your Julia code with colors and/or a surrounding box. There are currently five different themes: Two themes which are very similar to the style of the official Julia online documentation, two black and white themes and one dark theme.

2 License

manual.pdf

Copyright 2018 GitHub user wg030

This work may be distributed and/or modified under the conditions of the L^AT_EX Project Public License, either version 1.3 of this license or (at your option) any later version. The latest version of this license is in <http://www.latex-project.org/lppl.txt> and version 1.3 or later is part of all distributions of L^AT_EX version 2005/12/01 or later.

This work has the LPPL maintenance status 'maintained'.

The Current Maintainer of this work is GitHub user wg030.

This work consists of the files

jrcode.sty, *createlists.jl*, *testfile1.jl*, *testfile2.jl*, and *manual.pdf*.

3 Package Options

3.1 Current Package Options

autoload=*(true | false)* **(default: true)**

Load the jlcode style automatically when including the package.

This option is recommended if you just want to display Julia code with the listings package in your document. However if you also want to display code from different programming languages, you should set this option to false because you are likely to experience ugly interferences otherwise.

theme=*(string)* **(default: default)**

Choose a theme that defines the colors for the elements of your code as well as the appearance of a code box if desired.

Currently there exist the following five themes:

default, *default-plain*, *grayscale*, *grayscale-plain* and *darkbeamer*.

usecourier=*(true | false)* **(default: true)**

Use Courier as the typewriter font in your document.

This option is recommended unless you want to have a different font for your code examples. Note that only the typewriter family is affected here and you must load any font package before the jlcode package since otherwise there will be no effect with this option.

Moreover if you set this option to *false*, make sure your font supports bold typewriter font, like Courier does, because otherwise the keywords will not be displayed in bold, of course.

charsperline=*(positive integer)* **(default: 80)**

Control the width of the code box.

Use this option in order to specify the exact number of characters per line that can fit into the code box.

3.2 Obsolete Package Options

usecolors= \langle *true* | *false* \rangle (default: *true*)

This option is obsolete and will throw an error since version 5.0.
Use the *theme* option instead.

nocolors= \langle *true* | *false* \rangle (default: *false*)

This option is obsolete and will throw an error since version 5.0.
Use the *theme* option instead.

usebox= \langle *true* | *false* \rangle (default: *true*)

This option is obsolete and will throw an error since version 5.0.
Use the *theme* option instead.

nobox= \langle *true* | *false* \rangle (default: *false*)

This option is obsolete and will throw an error since version 5.0.
Use the *theme* option instead.

4 How to Insert Code in Your Document

4.1 Inserting Code with `autoload=true`

Command for Loading the Package:

```
\usepackage[autoload=true]{jlcode}
```

In-line Code Snippets:

```
\jlinl{@time sort(myarr) # no modification}
```

Note:

You must decode the following four characters

`{` `}` `%` `\` as `\{` `\}` `\%` `\\`
if you want to display them with the `\jlinl` command.

Display Code:

```
\begin{jllisting}  
# some julia code  
println( "Here we go with Julia!")  
\end{jllisting}
```

Listings for Standalone Files:

```
\jlinputlisting{filename.jl}
```

4.2 Inserting Code with `autoload=false`

Command for Loading the Package:

```
\usepackage[autoload=false]{jlcode}
```

In-line Code Snippets:

```
\jlinl{@time sort(myarr) # no modification}
```

Note:

You must decode the following four characters

`{` `}` `%` `\` as `\{` `\}` `\%` `\\`

if you want to display them with the `\jlinl` command.

Display Code:

```
\begin{jllisting}[language=julia , style=jlcodestyle]  
# some julia code  
println( "Here we go with Julia!")  
\end{jllisting}
```

Listings for Standalone Files:

```
\jlinputlisting[language=julia , style=jlcodestyle]{filename.jl}
```

5 How to Increase the Font Size

The `jlcode` package was designed in such a way that the size of the displayed code adjusts automatically to the font size of the current active font. As consequence of that you can simply increase (or decrease) the font size of your code as follows:

Changing the Font Size of In-line Code Snippets:

```
{\LARGE Quick efficiency check: \jlinl{@time sort(myarr)}}
```

Changing the Font Size of Display Code:

```
{\LARGE
\begin{jllisting}[language=julia , style=jlcodestyle]
# some julia code
println( "Here we go with Julia!")
\end{jllisting}
}
```

Changing the Font Size of Listings for Standalone Files:

```
{\LARGE \jlinputlisting{filename.jl}}
```

6 Themes

With jlcode version 5.0 themes were introduced into the package. A theme consists of a combination of colors for the elements of the code as well as a style for the appearance of the code box. The following five themes exist at the moment:

default:

```
\usepackage[theme=default]{jlcode}
```

This theme is the default theme. It uses the colors as well as the code box of the julia online documentation.

grayscale:

```
\usepackage[theme=grayscale]{jlcode}
```

This theme only uses black as color for the code elements and draws the same box as in the julia online documentation.

default-plain:

```
\usepackage[theme=default-plain]{jlcode}
```

This theme uses the colors of the julia online documentation, but does not draw a box.

grayscale-plain:

```
\usepackage[theme=grayscale-plain]{jlcode}
```

This theme only uses black as color for the code elements and does not draw a box.

darkbeamer:

```
\usepackage[theme=darkbeamer]{jlcode}
```

This theme is a dark theme which was designed by the GitHub user dietercastel. It was designed in such a way that it is ideally suited for the dark-beamer-theme.

6.1 Example with theme=default

```
#= A comment that consists of several lines.
The following code itself is rather useless unless you want
to test how Julia code is displayed by the jlcode package. =#

# This line will be my reference line, which will contain exactly 80 characters.

# This line is a comment containing operators like &, -, $ and %
# A comment with the German word "Übergrößengeschäft" (store for oversizes)
# This line contains some special unicode characters: €, α, γ, w², Δₓ, Ÿ, x̄, e
# A comment with some numbers: 424, 1.23, 0.2E-5, -9.9e+9, 1_001
# Mathematical characters that are Julia functions:
# |, |>, ~, x, ÷, €, ∄, ∃, ∄, ∘, √, ∛, ∩, ∪,
# ≈, ≠, ≠, ≡, ≠, ≤, ≥, ⊆, ⊇, ℤ, ℤ, ℤ, ℤ, √, √,
# Other mathematical symbols: ∇, ⊗, ⊕, ∥, ... , ... , ∴, ∴, ∴

# defining a useless testfunction
function Style_4th_Test(x, y)

    myver = v"2.00"
    mystr = "String: \"Übergrößengeschäft\", α, π, ∪, φ and the + operator."
    myset = Set{ [2, 9, 1_200, 2_500, 33]}
    x_in_myset = x ∈ myset
    myset{2} = myset ∪ Set{ [4, 8_000, 12, 33]}
    z1vec = rand{Int8, 3}
    z2vec = Array{Int8}(undef, 3)
    z2vec[1:2] = [x % y, y \ x]
    t = x % 2 == 0 ? x : x + 1
    t̃ = ~(t & x | y) ∨ y
    myβvar = t̃ & t $ t
    α = @time √0.3
    β^α = 3.2e+5^α
    myβvar = √0.12E-2 * β^α
    z2vec[3] = y^2 + 3.4x*y - (α + myβvar) * t/2
    z2vec = (z2vec + z1vec).^2
    if !(0.1 ≤ norm(z1vec') < norm(z2vec') + e + e ÷ pi + γ + φ)
        mystr = String{ mystr, " signed " }
        println( mystr )
        return true;
    elseif 3.2 ≥ norm(z2vec - z1vec) > 2.69
        if norm(z2vec - z1vec) ≠ 3.0
            println( String( "Error in ", myver, "!" ) )
        end
        return false;
    end
end

end
```

6.2 Example with theme=grayscale

```
#= A comment that consists of several lines.
The following code itself is rather useless unless you want
to test how Julia code is displayed by the jlcode package. =#

# This line will be my reference line, which will contain exactly 80 characters.

# This line is a comment containing operators like &, -, $ and %
# A comment with the German word "Übergrößengeschäft" (store for oversizes)
# This line contains some special unicode characters: €, α, γ, w², Δₓ, Ÿ, x̄, e
# A comment with some numbers: 424, 1.23, 0.2E-5, -9.9e+9, 1_001
# Mathematical characters that are Julia functions:
# |, |>, ~, x, ÷, €, ∄, ∃, ∄, ∘, √, ∛, ∩, ∪,
# ≈, ≠, ≠, ≡, ≠, ≤, ≥, ⊆, ⊇, ℤ, ℤ, ℤ, ℤ, √, √,
# Other mathematical symbols: ∇, ⊗, ⊕, ∥, ... , ... , ∴, ∴, ∴

# defining a useless testfunction
function Style_4th_Test(x, y)

    myver = v"2.00"
    mystr = "String: \"Übergrößengeschäft\", α, π, ∪, φ and the + operator."
    myset = Set{ [2, 9, 1_200, 2_500, 33]}
    x_in_myset = x ∈ myset
    myset{2} = myset ∪ Set{ [4, 8_000, 12, 33]}
    z1vec = rand{Int8, 3}
    z2vec = Array{Int8}(undef, 3)
    z2vec[1:2] = [x % y, y \ x]
    t = x % 2 == 0 ? x : x + 1
    t̃ = ~(t & x | y) ∨ y
    myβvar = t̃ & t $ t
    α = @time √0.3
    β^α = 3.2e+5^α
    myβvar = ∛0.12E-2 * β^α
    z2vec[3] = y^2 + 3.4x*y - (α + myβvar) * t/2
    z2vec = (z2vec + z1vec).^2
    if !(0.1 ≤ norm(z1vec') < norm(z2vec') + e + e ÷ pi + γ + φ)
        mystr = String( mystr, " signed ")
        println( mystr)
        return true;
    elseif 3.2 ≥ norm(z2vec - z1vec) > 2.69
        if norm(z2vec - z1vec) ≠ 3.0
            println( String( "Error in ", myver, "!"))
        end
        return false;
    end
end

end
```

6.3 Example with theme=default-plain

```
#= A comment that consists of several lines.
The following code itself is rather useless unless you want
to test how Julia code is displayed by the jlcode package. =#

# This line will be my reference line, which will contain exactly 80 characters.

# This line is a comment containing operators like &, -, $ and %
# A comment with the German word "Übergrößengeschäft" (store for oversizes)
# This line contains some special unicode characters: €, α, γ, w², Δₓ, Ÿ, x̄, e
# A comment with some numbers: 424, 1.23, 0.2E-5, -9.9e+9, 1_001
# Mathematical characters that are Julia functions:
# |, |>, ~, ×, ÷, ∈, ∉, ∃, ∄, °, √, ∛, ∩, ∪,
# ≈, ≠, ≠, ≡, ≠, ≤, ≥, ⊆, ⊇, ℤ, ℤ, ⊂, ⊃, ∖, ∙
# Other mathematical symbols: ∇, ⊗, ⊕, ||, ... , ... , ⋮, ⋯, ⋱

# defining a useless testfunction
function Style_4th_Test(x, y)

    myver = v"2.00"
    mystr = "String: \"Übergrößengeschäft\", α, π, ∪, φ and the + operator."
    myset = Set{ [2, 9, 1_200, 2_500, 33]}
    x_in_myset = x ∈ myset
    myset{2} = myset ∪ Set{ [4, 8_000, 12, 33]}
    z1vec = rand{Int8, 3}
    z2vec = Array{Int8}(undef, 3)
    z2vec[1:2] = [x % y, y \ x]
    t = x % 2 == 0 ? x : x + 1
    f̄ = ~(t & x | y) ∨ y
    myτvar = f̄ & t $ t
    α = @time √0.3
    β^α = 3.2e+5^α
    myβvar = √[3]{0.12E-2 * β^α}
    z2vec[3] = y^2 + 3.4x*y - (α + myβvar) * t/2
    z2vec = (z2vec + z1vec).^2
    if !(0.1 ≤ norm(z1vec') < norm(z2vec')) + e + e ÷ pi + γ + φ)
        mystr = String{ mystr, " signed "}
        println( mystr)
        return true;
    elseif 3.2 ≥ norm(z2vec - z1vec) > 2.69
        if norm(z2vec - z1vec) ≠ 3.0
            println( String{ "Error in ", myver, "!"})
        end
        return false;
    end
end
```

6.4 Example with theme=grayscale-plain

```

#= A comment that consists of several lines.
The following code itself is rather useless unless you want
to test how Julia code is displayed by the jlcode package. =#

# This line will be my reference line, which will contain exactly 80 characters.

# This line is a comment containing operators like &, -, $ and %
# A comment with the German word "Übergrößengeschäft" (store for oversizes)
# This line contains some special unicode characters: €, α, γ, w², Δₓ, Ĩ, x̃, e
# A comment with some numbers: 424, 1.23, 0.2E-5, -9.9e+9, 1_001
# Mathematical characters that are Julia functions:
# |, |>, ~, x, ÷, €, ∄, ∃, ∄, ∘, √, ∛, ∩, ∪,
# ≈, ≠, ≠, ≡, ≠, ≤, ≥, ⊆, ⊃, ⊆, ⊇, ⊆, ⊇, ∨, ∙
# Other mathematical symbols: ∇, ⊗, ⊕, ∥, …, …, …, …, …, …

# defining a useless testfunction
function Style_4th_Test(x, y)

    myver = v"2.00"
    mystr = "String: \"Übergrößengeschäft\", α, π, ∪, φ and the + operator."
    myset = Set{ [2, 9, 1_200, 2_500, 33]}
    x_in_myset = x ∈ myset
    myset{2} = myset ∪ Set{ [4, 8_000, 12, 33]}
    z1vec = rand{Int8, 3}
    z2vec = Array{Int8}(undef, 3)
    z2vec[1:2] = [x % y, y \ x]
    t = x % 2 == 0 ? x : x + 1
    t̃ = ~(t & x | y) ∨ y
    myβvar = t̃ & t $ t
    α = @time √0.3
    βᵅ = 3.2e+5ᵅ
    myβvar = √0.12E-2 * βᵅ
    z2vec[3] = y² + 3.4x*y - (α + myβvar) * t/2
    z2vec = (z2vec + z1vec).^2
    if !(0.1 ≤ norm(z1vec') < norm(z2vec') + e + e ÷ pi + γ + φ)
        mystr = String( mystr, " signed ")
        println( mystr)
        return true;
    elseif 3.2 ≥ norm(z2vec - z1vec) > 2.69
        if norm(z2vec - z1vec) ≠ 3.0
            println( String( "Error in ", myver, "!"))
        end
        return false;
    end
end

end

```

6.5 Example with theme=darkbeamer

```

1 #=: A comment that consists of several lines.
2 The following code itself is rather useless unless you want
3 to test how Julia code is displayed by the jlcode package. =#
4
5 # This line will be my reference line, which will contain exactly 80 characters.
6
7 # This line is a comment containing operators like &, -, $ and %
8 # A comment with the German word "Übergrößengeschäft" (store for oversizes)
9 # This line contains some special unicode characters: €, α, γ, w², Δx, ħ, x̄, e
10 # A comment with some numbers: 424, 1.23, 0.2E-5, -9.9e+9, 1_001
11 # Mathematical characters that are Julia functions:
12 # |, |>, ~, ×, ÷, ∈, ∉, ∋, ∌, ∘, √, ∛, ∩, ∪,
13 # ≈, ≇, ≠, ≡, ≢, ≤, ≥, ⊆, ⊃, ℤ, ℒ, ⅈ, ⅉ, ∞, ∙
14 # Other mathematical symbols: ∇, ⊗, ⊕, ‖, …, …, ⋮, ⋯, ⋱
15
16 # defining a useless testfunction
17 function Style_4th_Test(x, y)
18
19     myver = v"2.00"
20     mystr = "String: \"Übergrößengeschäft\", α, π, ∪, φ and the + operator."
21     myset = Set{Int}([2, 9, 1_200, 2_500, 33])
22     x_in_myset = x ∈ myset
23     myset{2} = myset ∪ Set{Int}([4, 8_000, 12, 33])
24     z1vec = rand{Int8, 3}
25     z2vec = Array{Int8}(undef, 3)
26     z2vec[1:2] = [x % y, y \ x]
27     t = x % 2 == 0 ? x : x + 1
28     t̃ = ~(t & x | y) ∨ y
29     mytvar = t̃ & t $ t
30     α = @time √0.3
31     β^α = 3.2e+5^α
32     myβvar = ∛0.12E-2 * β^α
33     z2vec[3] = y^2 + 3.4x*y - (α + myβvar) * t/2
34     z2vec = (z2vec + z1vec).^2
35     if !(0.1 ≤ norm(z1vec') < norm(z2vec') + e + e ÷ pi + γ + φ)
36         mystr = String(mystr, " signed ")
37         println(mystr)
38         return true;
39     elseif 3.2 ≥ norm(z2vec - z1vec) > 2.69
40         if norm(z2vec - z1vec) ≠ 3.0
41             println(String("Error in ", myver, "!"))
42         end
43         return false;
44     end
45 end
46 end

```

7 Known Manageable Issues

The following Julia code presents the known issues that can appear due to the nature of the listings package. Right now the jlcode package is not able to handle these cases. However the issues can be fixed by the user himself.

Output Without Fixing the Issues:

```
function KnownManageableIssues(x, y)

    # KNOWN MANAGEABLE ISSUES:

    # identifier name with a number that follows
    # directly behind a special unicode character:
    myβ2ndvar = 2 * 0.12E-2 * xy

    # identifier name, which contains a γ, π or φ:
    myφvar+ = sqrt(2)
    approx4π = 3.142

    # numbers in E-notation without using a + sign:
    evar = 3.99e400
    evar2 = 3.99E400

    # single characters
    mychar = 'W'
    mychar(2) = '€'
    mychar(3) = 'κ'

end
```

Commands for Fixing the Issues:

```
\addlitjlbase{myβ2ndvar}{my$\beta$2ndvar}{9}
\addlitjlbase{myφvar+}{my$\phi$var$\scriptstyle {}_{+}}{7}
\addlitjlbase{approx4π}{approx4$\pi$}{8}
\addlitjlbase{löwe}{l{"o}we}{4}
\addlitjlbase{checkin€}{checkin\euro}{8}
\addlitjlstrnum{e400}{e400}{4}
\addlitjlstrnum{E400}{E400}{4}
\addlitjlstrnum{'W'}{\textquotesingle W\textquotesingle}{3}
\addlitjlstrnum{'€'}{\textquotesingle \euro\textquotesingle}{3}
\addlitjlstrnum{'κ'}{\textquotesingle $\varkappa$\textquotesingle}{3}
```

Output After Fixing the Issues:

```
function KnownManageableIssues(x, y)

  # KNOWN MANAGEABLE ISSUES:

  # identifier name with a number that follows
  # directly behind a special unicode character:
  my $\beta$ 2ndvar = 2 * 0.12E-2 * xy

  # identifier name, which contains a  $\gamma$ ,  $\pi$  or  $\phi$ :
  my $\phi$ var+ = sqrt(2)
  approx4 $\pi$  = 3.142

  # numbers in E-notation without using a + sign:
  evar = 3.99e400
  evar2 = 3.99E400

  # single characters
  mychar = 'w'
  mychar(2) = '€'
  mychar(3) = 'x'

end
```