MANUAL for

THE *jlcode* PACKAGE Version 4.0

February 27, 2020

(Version 1.0: January 25, 2018)

https://github.com/wg030/jlcode

Contents

1 Introduction

The jlcode package (*jlcode.sty*) provides a language definition for the programming language Julia for the listings package as well as a style definition. 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 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 nice looking style was defined, too, which will help you to highlight your Julia code with colors and a surrounding box. The style is very similiar to the style of the official Julia online documentation, but it can be turned off partly or even completely.

2 License

3 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.

usecolors=⟨*true* | *false*⟩ **(default: *true*)**

Activate the coloring of keywords, literals, built-ins and macros.
Use this option if you want important aspects of your code to be highlighted with some nice colors.

nocolors=⟨*true* | *false*⟩ **(default: *false*)**

The opposite of usecolors. Deactivate the coloring of keywords, literals, built-ins and macros.
Use this option if you want all parts of your code to be printed in black.

usebox=⟨*true* | *false*⟩ **(default: *true*)**

Activate the code box.
Use this option if you want to display a light grey code box.

nobox=⟨*true* | *false*⟩ **(default: *false*)**

The opposite of usebox. Deactivate the code box.
Use this option if you want your code de be displayed without a code box.

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

Control the width of code box.
Use this option in order to specifiy the exact number of characters per line that can fit into the codebox. If you use the nobox option, this option is without any effect.

4 How to Insert Code with autoload=true

Command for Loading the Package:

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

In-line Code Snippets:

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

Diplay Code:

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

Listings for Standalone Files:

```
\lstinputlisting{filename.jl}
```

5 How to Insert Code with autoload=false

Command for Loading the Package:

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

In-line Code Snippets:

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

Diplay Code:

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

Listings for Standalone Files:

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

6 Example with usecolors=true and usebox=true

Command for Loading the Package:

```
\usepackage[usecolors,usebox]{jlcode}
```

Output of a Useless but Diverse Code Example:

```julia
#= 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, ϔ, x̃, ℯ
# A comment with some numbers: 424, 1.23, 0.2E-5, -9.9e+9, 1_001
# Mathematical characters that are Julia functions:
# |, |>, ~, ×, ÷, ∈, ∉, ∋, ∌, ∘, √, ∛, ∩, ∪,
# ≈, ≉, ≠, ≡, ≢, ≤, ≥, ⊆, ⊇, ⊈, ⊉, ⊊, ⊋, ⊻, ⋅
# Other mathematical symbols: ∇, ⊗, ⊕, ‖, …, ⋯, ⋮, ⋱, ⋰

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

    myver = v"2.00"
    mystr = "String: \"Übergrößengeschäft\", α, π, ∪, φ̇  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
    t̄ = ~(t & x | y) ⊻ y
    my÷̂var = t̄ & t $ t
    α = @time √0.3
    βᵅ = 3.2e+5^α
    myβvar = ∛0.12E-2 * βᵅ
    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
```

7 Example with nocolors=true and usebox=true

Command for Loading the Package:

```
\usepackage[nocolors,usebox]{jlcode}
```

Output of a Useless but Diverse Code Example:

```
#= 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, ϔ, x̃, ℯ
# A comment with some numbers: 424, 1.23, 0.2E−5, −9.9e+9, 1_001
# Mathematical characters that are Julia functions:
# |, |>, ~, ×, ÷, ∈, ∉, ∋, ∌, ∘, √, ∛, ∩, ∪,
# ≈, ≉, ≠, ≡, ≢, ≤, ≥, ⊆, ⊇, ⊈, ⊉, ⊊, ⊋, ⊻, ⋅
# Other mathematical symbols: ∇, ⊗, ⊕, ‖, …, ⋯, ⋮, ⋱, ⋰

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

    myver = v"2.00"
    mystr = "String: \"Übergrößengeschäft\", α, π, ∪, φ̇  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
    t̄ = ~(t & x | y) ⊻ y
    myτ̂var = t̄ & t $ t
    α = @time √0.3
    βᵅ = 3.2e+5^α
    myβvar = ∛0.12E−2 ⋆ βᵅ
    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
```

8 Example with usecolors=true and nobox=true

Command for Loading the Package:

```
\usepackage[usecolors,nobox]{jlcode}
```

Output of a Useless but Diverse Code Example:

```
#= 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, Ϋ, x̃, e
# A comment with some numbers: 424, 1.23, 0.2E-5, -9.9e+9, 1_001
# Mathematical characters that are Julia functions:
# |, |>, ~, ×, ÷, ∈, ∉, ∋, ∌, ∘, √, ∛, ∩, ∪,
# ≈, ≉, ≠, ≡, ≢, ≤, ≥, ⊆, ⊇, ⊈, ⊉, ⊊, ⊋, ⊻, ⋅
# Other mathematical symbols: ∇, ⊗, ⊕, ‖, …, ⋯, ⋮, ⋱, ⋰

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

    myver = v"2.00"
    mystr = "String: \"Übergrößengeschäft\", α, π, ∪, φ̇  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
    t̄ = ~(t & x | y) ⊻ y
    myτ̂var = t̄ & t $ t
    α = @time √0.3
    βᵅ = 3.2e+5^α
    myβvar = ∛0.12E-2 * βᵅ
    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
```

9 Example with nocolors=true and nobox=true

Command for Loading the Package:

```
\usepackage[nocolors,nobox]{jlcode}
```

Output of a Useless but Diverse Code Example:

```
#= 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, Ϋ, x̃, ℯ
# A comment with some numbers: 424, 1.23, 0.2E-5, -9.9e+9, 1_001
# Mathematical characters that are Julia functions:
# |, |>, ~, ×, ÷, ∈, ∉, ∋, ∌, ∘, √, ∛, ∩, ∪,
# ≈, ≉, ≠, ≡, ≢, ≤, ≥, ⊆, ⊇, ⊈, ⊉, ⊊, ⊋, ⊻, ⋅
# Other mathematical symbols: ∇, ⊗, ⊕, ‖, …, ⋯, ⋮, ⋱, ⋰

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

    myver = v"2.00"
    mystr = "String: \"Übergrößengeschäft\", α, π, ∪, φ̇  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
    t̄ = ~(t & x | y) ⊻ y
    myτ̂var = t̄ & t $ t
    α = @time √0.3
    βᵅ = 3.2e+5^α
    myβvar = ∛0.12E-2 * βᵅ
    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
```

10 Known Managable 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 theses 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}
\addlitjlstring{e400}{e400}{4}
\addlitjlstring{E400}{E400}{4}
\addlitjlstring{'W'}{\textquotesingle W\textquotesingle}{3}
\addlitjlstring{'€'}{\textquotesingle \euro\textquotesingle}{3}
\addlitjlstring{'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:
  myβ2ndvar = 2 * 0.12E-2 * x^y

  # 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
  evar_2 = 3.99E400

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

end
```