MANUAL for

THE *jlcode* PACKAGE Version 6.1

May 03, 2022

(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 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 (*createkwlists.jl*) and then included in the language definition of this package.

Since version 6.0 you cannot only compile your document with pdftex to get the desired results, but you can compile your document with luatex or xetex, too. There is no need to do something special when compiling with luatex or xetex either. The only thing you must make sure is that you do not forget to download the JuliaMono font files, which were created by the GitHub User cormullion.

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 similiar to the style of the official Julia online documentation, two black and white themes and one dark theme.

2 License

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 other programming languages, you should set this option to false because you are likely to experience ugly interferences otherwise.

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

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

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

Use Courier as the typewriter font in your document when compiling with pdftex respectively JuliaMono when compiling with luatex or xetex.
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 typpwriter font, like Courier and JuliaMono do, because otherwise the keywords will not be displayed in bold, of course.

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

Add line numbers to the code box.
Use this option if you want to number the lines of your code.

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*.

3.2 Obsolete Package Options

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

This option is obsolete and will throw an error since version 5.0.
Since version 6.0 the new name of this option is *defaultmonofont*.

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

This option is obsolete and will throw an error since version 5.0.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

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

This option is obsolete and will throw an error since version 5.0.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

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

This option is obsolete and will throw an error since version 5.0.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

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

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

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

This option is obsolete and will throw an error since version 6.0.
The option has been renamed to *defaultmonofont*.

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.

Diplay 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.

Diplay Code:

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

Listings for Standalone Files:

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

5 How to Create Labels and Captions for Your Code Blocks

In order to create captions and/or labels for your code blocks you can make use of the possibilities that the interface of the listing package offers you as follows:

Creating Labels and Captions for Diplay Code Listings:

```
\begin{jllisting}[caption={My Code}, label=mylabel]
# some julia code
println( "Here we go with Julia!")
\end{jllisting}
```

Creating Labels and Captions for Listings of Standalone Files:

```
\jlinputlisting[caption={My Code}, label=mylabel]{filename.jl}
```

6 How to Increase the Font Size

The jlcode package was desgined in such a way that the size of the displayed code adjusts automatically to the font size of the current active font. As a 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 Diplay Code:

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

Changing the Font Size of Listings for Standalone Files:

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

7 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 desgined in such a way that it is ideally suited for the dark-beamer-theme.

7.1 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, 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, ∪, 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]
    reverse!(z1vec)
    t = x % 2 == 0 ? x : x + 1
    t̄ = ~(t & x | y) ⊻ y
    my⃗var = t̄ & 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
```

7.2 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, 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, ∪, 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]
    reverse!(z1vec)
    t = x % 2 == 0 ? x : x + 1
    t̄ = ~(t & x | y) ⩴ y
    my⃗var = t̄ & 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
```

7.3 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, 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]
    reverse!(z1vec)
    t = x % 2 == 0 ? x : x + 1
    t̄ = ~(t & x | y) ⩴ y
    my𝜏var = t̄ & 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
```

7.4 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, 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]
    reverse!(z1vec)
    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
```

7.5 Example with theme=darkbeamer

```
#= 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, ŷ, 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, ∪, 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]
    reverse!(z1vec)
    t = x % 2 == 0 ? x : x + 1
    t̃ = ~(t & x | y) ⊻ y
    my̆var = 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
```

8 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 theses cases automatically. However the issues can be fixed manually by the user himself using the commands \addlitjlstrnum, \addlitjlbase, \addlitjlmacros and \addlitjlfunctions. These commands should be added to the preample of your latex document right after the jlcode package is loaded. Here are a few typical examples of issues that can be fixed manually:

Output Without Fixing the Issues:

```julia
# KNOWN MANAGEABLE ISSUES:

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


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


# calling self defined macros
@spellcheck("fukc")


# self defined functions ending with '!'
function changesig!( A)
    A .= -A
end




# KNOWN MANAGEABLE ISSUES (pdftex ENGINE ONLY):

# 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
```

Commands for Fixing the Issues:

```
\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}
\addlitjlbase{myβ2ndvar}{my$\beta$2ndvar}{9}
\addlitjlbase{myϕvar₊}{my$\phi$var${\scriptstyle {}_{+}}$}{7}
\addlitjlbase{approx4π}{approx4$\pi$}{8}
\addlitjlmacros{@spellcheck}{@spellcheck}{11}
\addlitjlfunctions{changesig!}{changesig!}{10}
```

Output After Fixing the Issues:

```
# KNOWN MANAGEABLE ISSUES:

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


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


# calling self defined macros
@spellcheck("fukc")


# self defined functions ending with '!'
function changesig!( A)
    A .= -A
end




# KNOWN MANAGEABLE ISSUES (pdftex ENGINE ONLY):

# 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
```