1 JuliaPro

2 (v0.5.0.4)

3 Installation Manual and Quickstart Guide

3.1 Contents

4 1. Objective

This guide details the installation procedure and usage of the base JuliaPro package and associated add-on packages for JuliaPro's Juno IDE, as well as the JuliaInXL package.

5 2. Prerequisites

- An appropriate version of Mac OSX or macOS
- Yosemite, El Capitan, Sierra
- Julia 0.5.x (bundled with the base JuliaPro installer)
- 5 GB of disk space

5.1 2.1 Prerequisites for use of IJulia, PyCall and PyPlot

To make use of the IJulia package for execution of Jupyter notebooks as well as the PyPlot package for producing plots via Python's matplotlib package, a working installation of Python must be present on the system as well as installations of the `jupyter`, `matplotlib`, and `ipywidgets` packages.

5.2 2.1.1 Installing Python and packages via Homebrew

One method of installing Python and associated packages is via the Homebrew package manager. To install Homebrew, first launch a Terminal application. To launch a Terminal window from within OSX, first click on the Launchpad icon in the Dock on your desktop, then click the "Other" icon, and select the "Terminal" application.

From within the Terminal application, installation of Homebrew can be performed via the following command:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/
Homebrew/install/master/install)"
```

Installation of Homebrew may require the user to have `sudo` privileges for administrative installations.

To install a copy of Python via Homebrew, execute the following command:

```
brew install python
```

This command will create a link to the `python` executable at `/usr/local/bin/python`.

The copy of Python installed via Homebrew includes a copy of the `pip` package manager for installing Python packages.

To install the Jupyter, Matplotlib and ipywidgets packages, execute the following command:

```
pip install jupyter matplotlib ipywidgets
```

A link to the installed `jupyter` executable is present at `/usr/local/bin/jupyter`.

As part of executing the JuliaPro installer, two entries are added by default to the `ENV` dictionary from within the `juliarc.jl` file:

- `ENV["PYTHON"]=/usr/local/bin/python`
- `ENV["JUPYTER"]=/usr/local/bin/jupyter`

Configuration of the `juliarc.jl` file is described in more detail in the following section.

5.3 2.1.2 Configuring IJulia, PyCall and PyPlot for use with other Python distributions

To configure various Python interop packages for use with other Python distributions requires the manual configuration of the PyCall and IJulia packages for use with those Python installations.

Within the file `/Applications/JuliaPro-0.5.0.4.app/Contents/Resources/julia/Contents/Resources/julia/etc/julia/juliarc.jl` the `ENV` dictionary containing inherited system environment variables is modified to add entries for `ENV["PYTHON"]` and `ENV["JUPYTER"]`. By default, these entries take on the values `/usr/local/bin/python` and `/usr/local/bin/jupyter` respectively.

To point the PyCall and IJulia packages at the `python` and `jupyter` executables for your installation of Python, perform the following steps:

1. Create two system level environment variables, `PYTHON` and `JUPYTER` either in your Terminal shell or within your .bash_profile. These variables should be assigned values that point to the `python` and `jupyter` executable binaries within your Python installation.
2. Launch JuliaPro-0.5.0.4 from the terminal as follows:

```
$/Applications/JuliaPro-0.5.0.4.app/Contents/Resources/julia/
Contents/Resources/julia/bin/julia
```

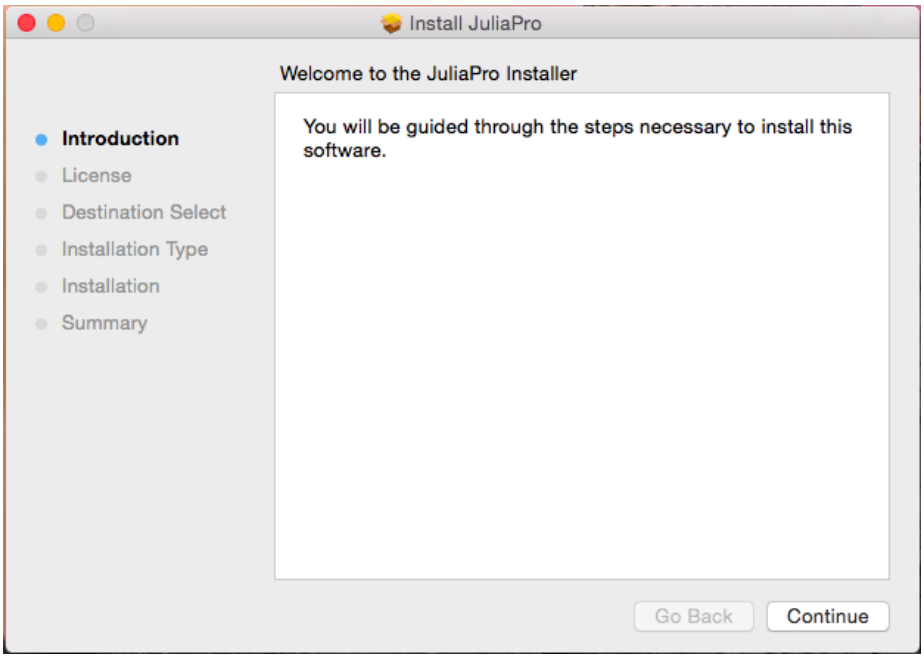
3. Rebuild the PyCall package: `julia julia> Pkg.build("PyCall")`
4. Rebuild the IJulia package: (NOTE: This currently fails on my machine) `julia julia> Pkg.build("IJulia")`

At this point, one can load the IJulia, PyCall or PyPlot packages to create a Jupyter session, load a Python module, or create a matplotlib plot from within a Jupyter notebook using PyPlot.

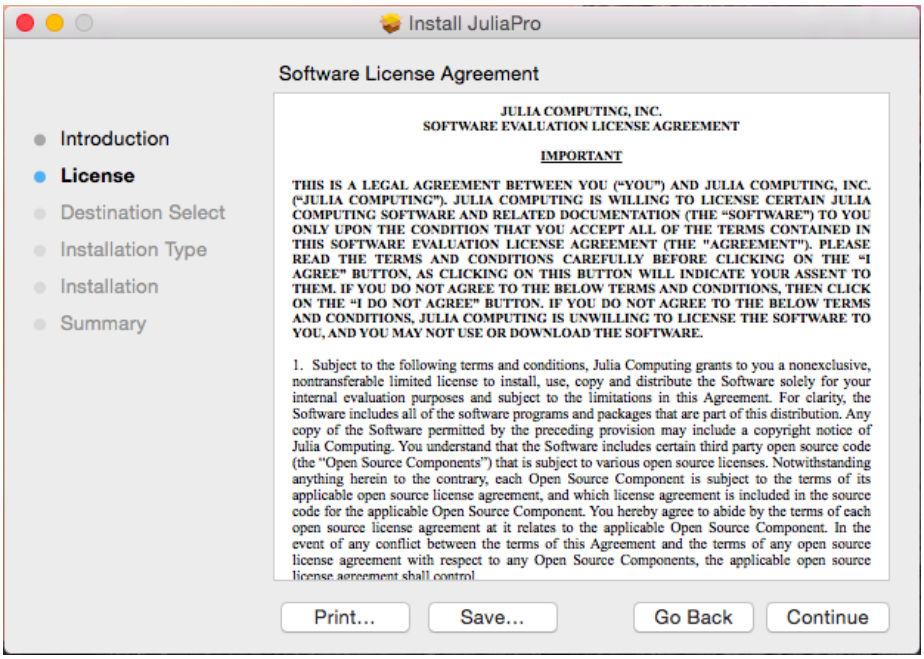
6 3. Installing JuliaPro

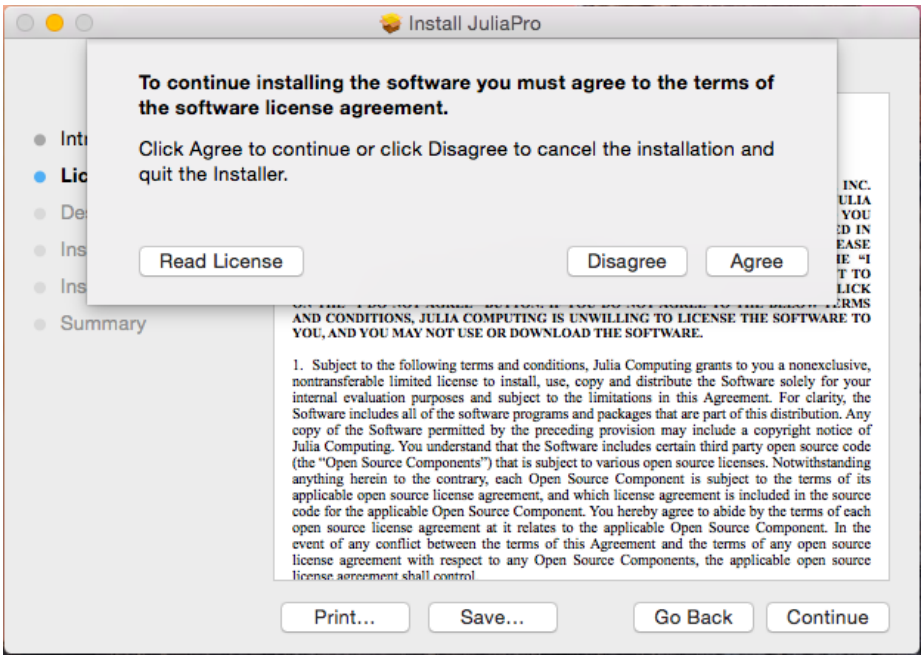
Before installing the latest version of JuliaPro for Mac, you must first uninstall any existing version of JuliaPro for Mac that is installed on your machine.

Once the requirements are met, you can start the JuliaPro installation using the installer provided.

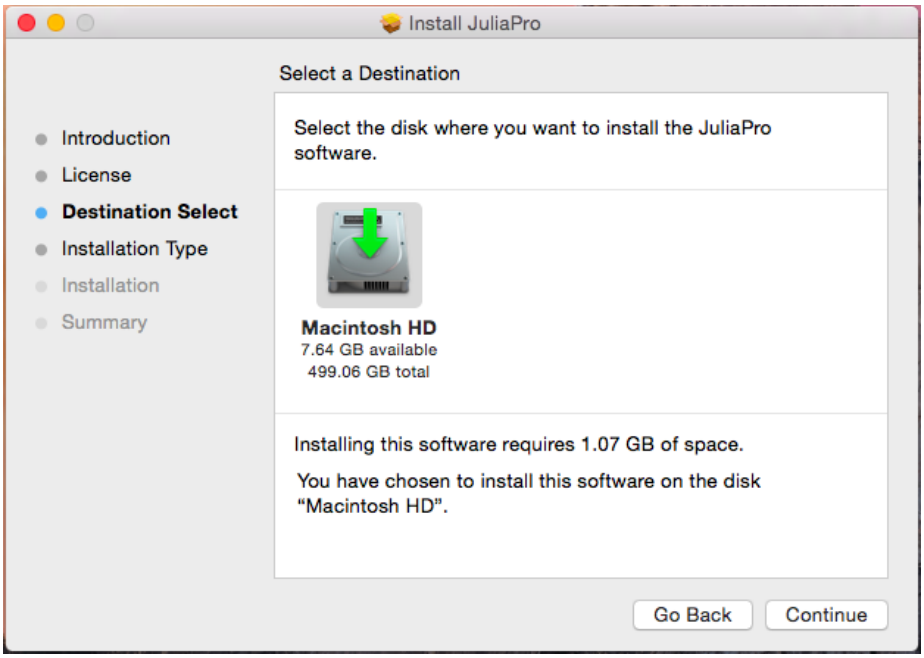


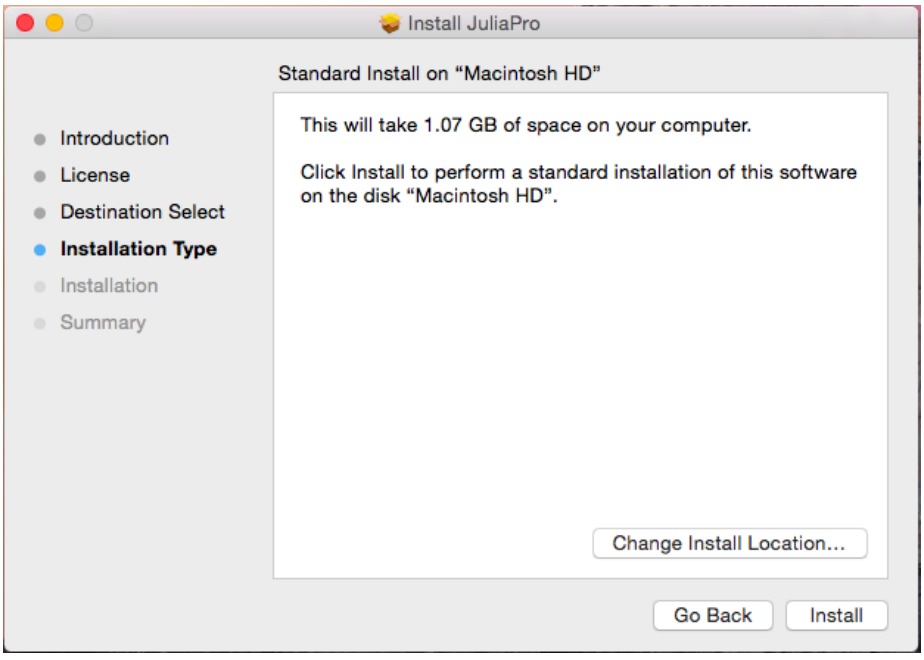
Upon starting the installation, you will be presented with the JuliaPro Software License Agreement. After reading through the terms mentioned in the agreement, click "Agree" if you accept the terms of the license and proceed with the installation.



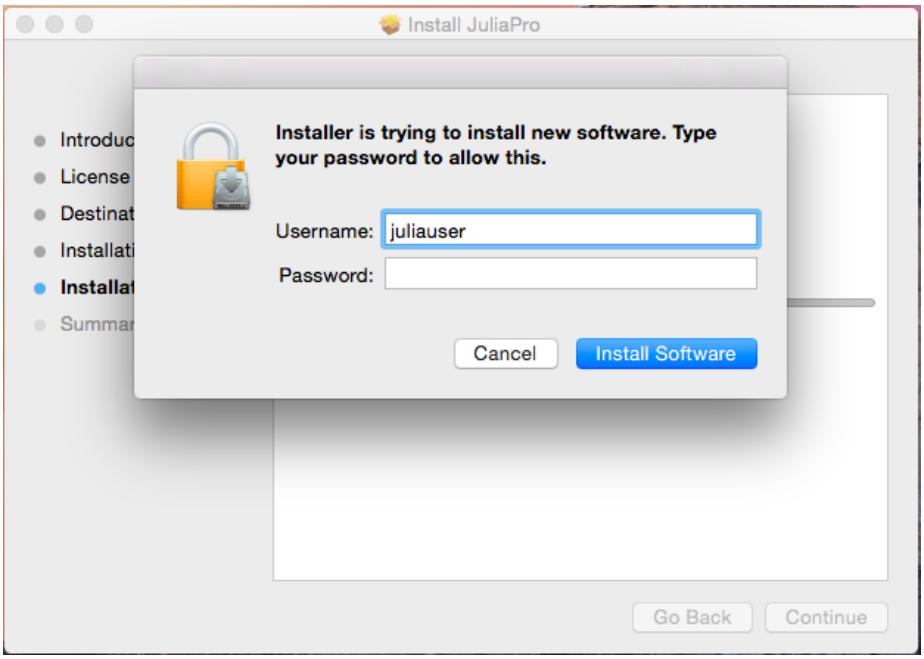


After accepting the license agreement, you can select the location where JuliaPro will be installed.

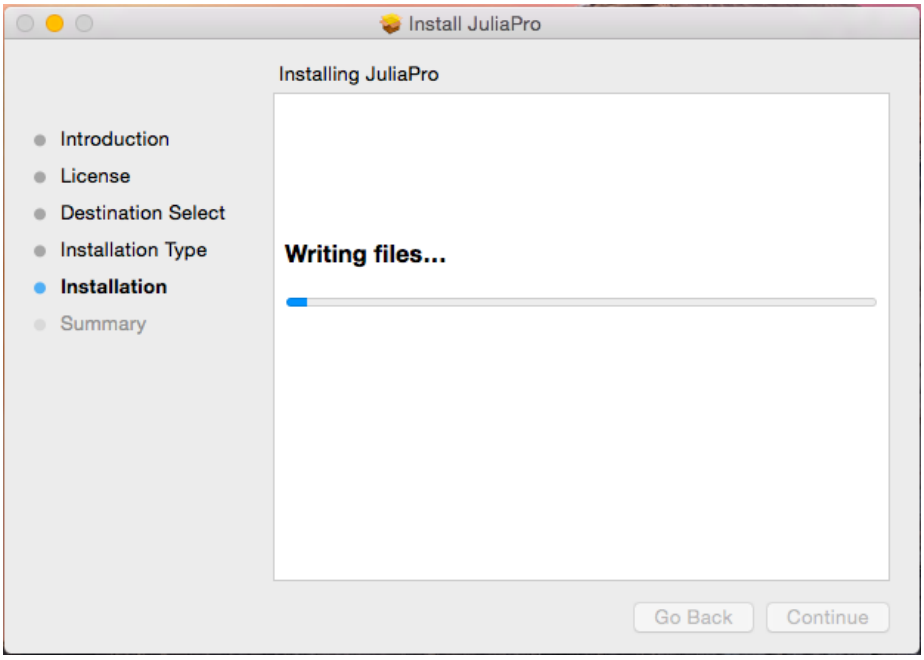




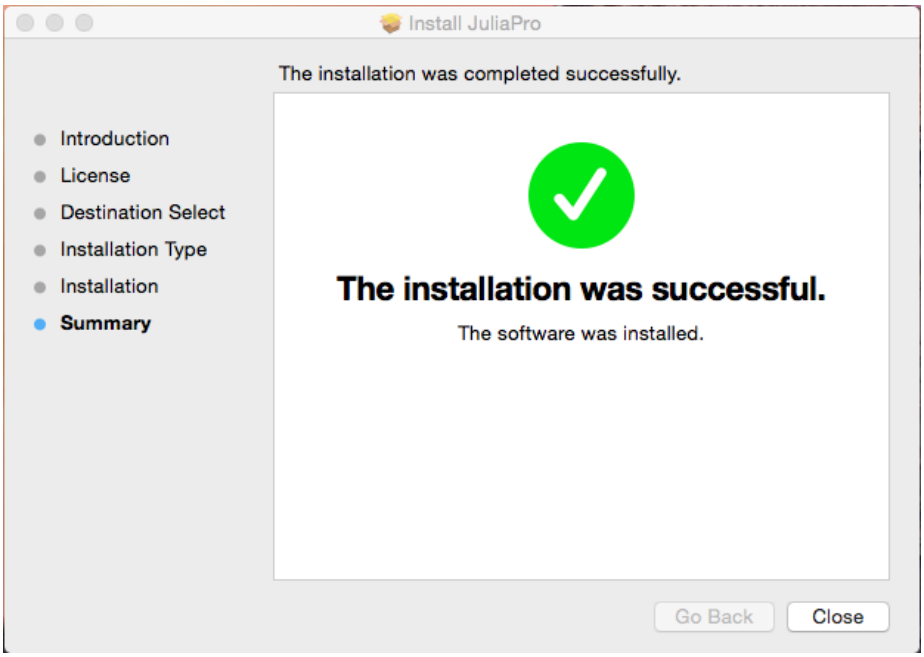
Depending on your current account security settings, you might need to provide your current user account password to proceed with the installation.



The installer will then proceed to install JuliaPro in the chosen location.



Upon completion of the installer, press close to exit the installer.



7 4. Using the JuliaPro Command Prompt

To launch a JuliaPro Command Prompt within a Terminal window, one needs to launch the `julia` binary included within the JuliaPro distribution.

To launch a Terminal window from within OSX, first click on the Launchpad icon in the Dock on your desktop, then click the "Other" icon, and select the "Terminal" application.

From within an executing Terminal application, the `julia` binary can be executed via the following command:

```
$ /Applications/JuliaPro-0.5.0.4.app/Contents/Resources/julia/
Contents/Resources/julia/bin/julia
```

Upon launching the `julia` binary, a Julia REPL (read-eval-print-loop) will be available to execute Julia commands interactively.

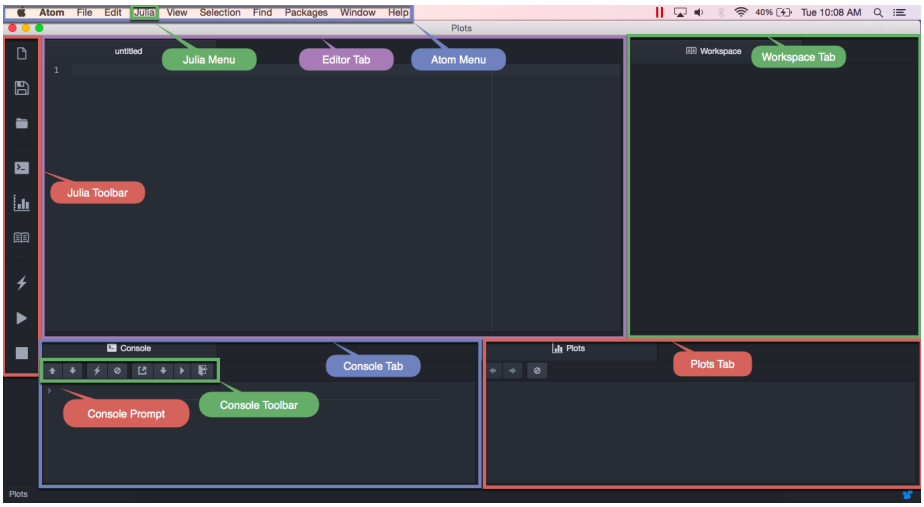
To inspect all of the packages currently installed as part of JuliaPro, execute the `Pkg.status()` command at the julia command prompt.

8 5. Using Juno for JuliaPro

8.1 5.1 Launching Juno for JuliaPro

From within the LaunchPad window, Double-Click on the JuliaPro icon to start a Juno session.

Upon initially launching Juno, you will be presented with the following window.



Juno is a full-featured and easily extensible integrated development environment based on the Atom editor.

The previous screenshot shows the most important aspects of the Juno environment for working with Julia code:

- The Editor Tab - A tab that is primarily used for developing longer segments of Julia code, as well as code that will be saved to a file. Multiple editor tabs can be opened within a single Juno session to develop multiple Julia files.
- The Console Tab - The tab that is primarily used for executing Julia commands interactively and displaying textual results from those commands. The Console Tab also includes the Console Toolbar that includes various buttons for control of the Console, as well as for debugging code via Gallium within Juno.
- The Plots Tab - The tab used for display of visualizations produced by supported plotting packages.
- The Workspace Tab - The tab used for displaying the values of variables and functions that are in the current Julia scope.
- The Julia Toolbar - The primary toolbar containing buttons for controlling aspects of the use of Julia from within Juno.
- The Julia Menu - An entry in the the Atom Menu that is specific to control of Julia from within Juno.
- The Atom Command Palette (not shown in the screenshot above) - A pop-up window that allows for searching all commands available from within an Atom/Juno session, including "julia" specific commands.

Subseqeuent sections will describe the functionality included in each of these components of Juno for JuliaPro. But first, we will take you through a couple of common and basic of workflows when getting started with Julia.

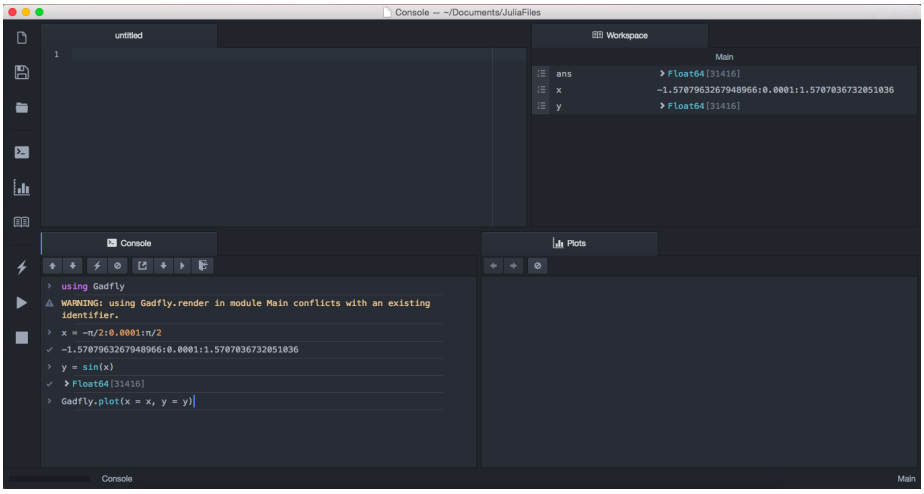
8.2 5.2. Getting started with Juno for JuliaPro

When getting started with the use of Julia in the Atom IDE, there are four primary panes where users will spend the majority of their time and effort:

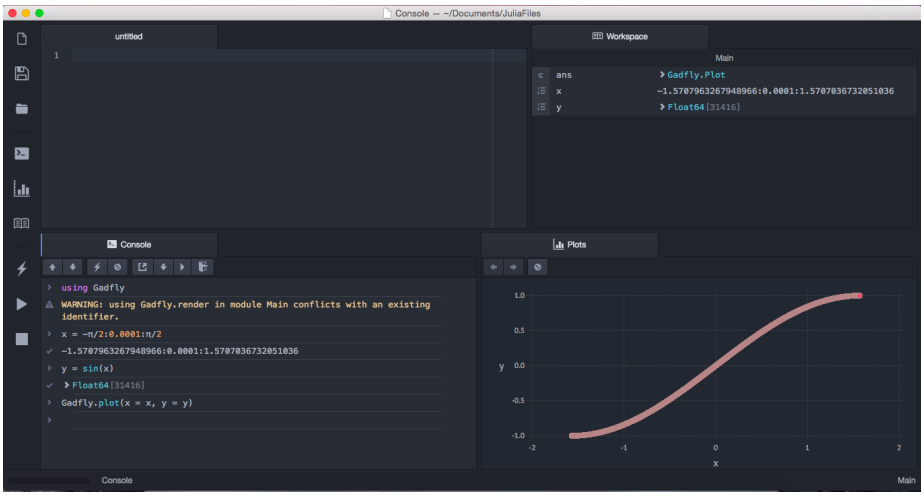
- The Editor Tab
- The Console Tab
- The Plots Tab
- The Workspace Tab

Within the Editor tab, users can create, edit, open and save Julia source code files. Juno also allows for code to be executed, and associated results displayed, directly inline within the Editor tab. The Console tab provides an interactive Julia command prompt for immediate command execution, and the Plots tab can display visualizations produced by either Gadfly.jl (included with JuliaPro) or Plots.jl. The Workspace tab provides a summary window of all variables and functions defined in the execution scope of the current Julia session.

As a first set of operations, one can execute expressions, load a package, define variables, and create a plot all from the Console tab. The example below shows an example of loading Julia's Gadfly package, defining independent and dependent variable vectors, and executing Gadfly's plot command.

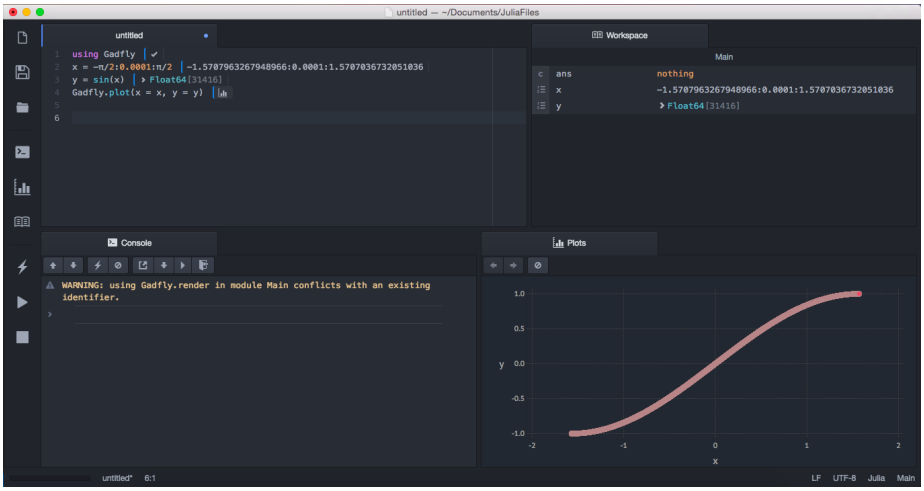


A resulting plot is displayed within the Plots tab.



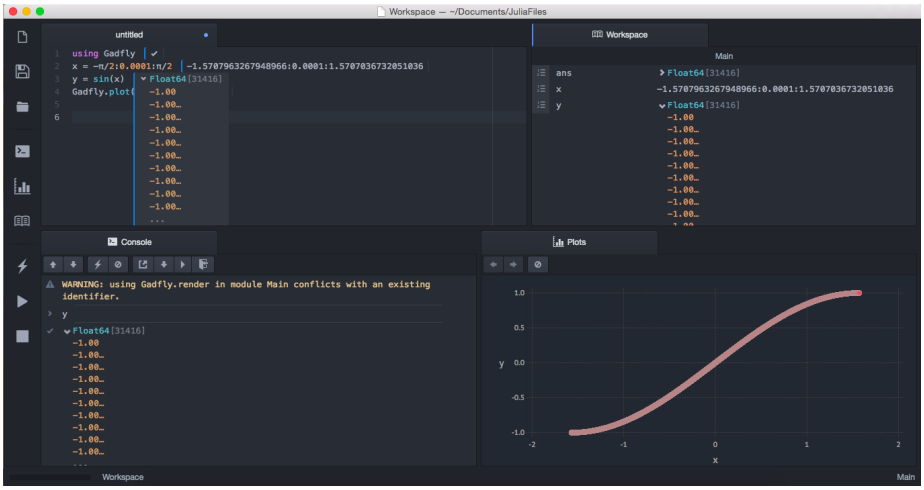
While some users' preferred workflow involves developing and executing short interactive snippets of code in the Console tab, and typing longer segments of code within the Editor tab, Juno also provides functionality for executing code directly within the Editor tab.

Below is an example Juno session showing the same set of operations above being executing in-line within the Editor tab. To execute commands directly within the Editor tab, place the cursor within the line to be executed and press the key combination Shift+Enter.



The contents of any assigned variable can be viewed either the Workspace, the Console, or directly within the Editor for any statements that have been directly executed in the Editor. For any Julia array, DataFrame or custom type that has been defined, the values assigned to a variable can be viewed in an expanded form by clicking on the > character that is displayed to the left of the summary

description in the Workspace, the Console, or in the Editor.



Any operations executed from within the Editor tab are also within the scope of the current Console tab, and the results of any assignment or defintion are viewable in the Workspace tab.

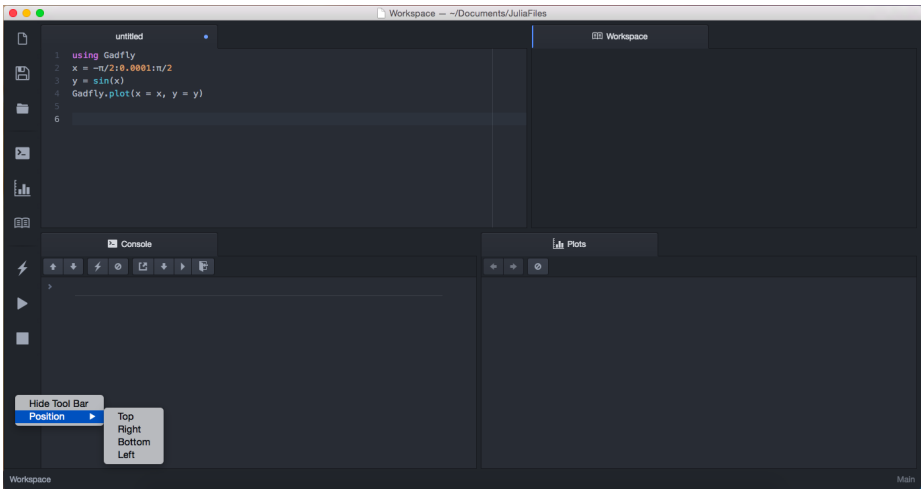
8.3 5.2.1 Using the Julia Toolbar

The section describes the contents and use of the Julia Toolbar

8.4 5.2.1.1 Toolbar Location

The Julia Toolbar defaults to being displayed on the left hand side of the editor window. Both the location of the toolbar and visibility of the toolbar are configurable.

To change either the location or the visibility of the toolbar, a right-click anywhere on the toolbar displays a pop-up window that allows for either hiding the toolbar, or selecting the edge of the Atom window along which the editor is to be displayed.

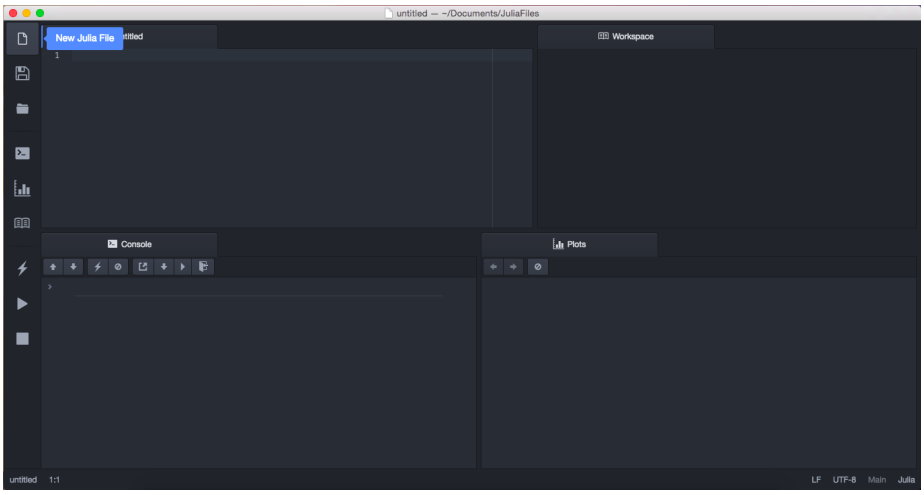


8.5 5.2.1.2 Toolbar Contents

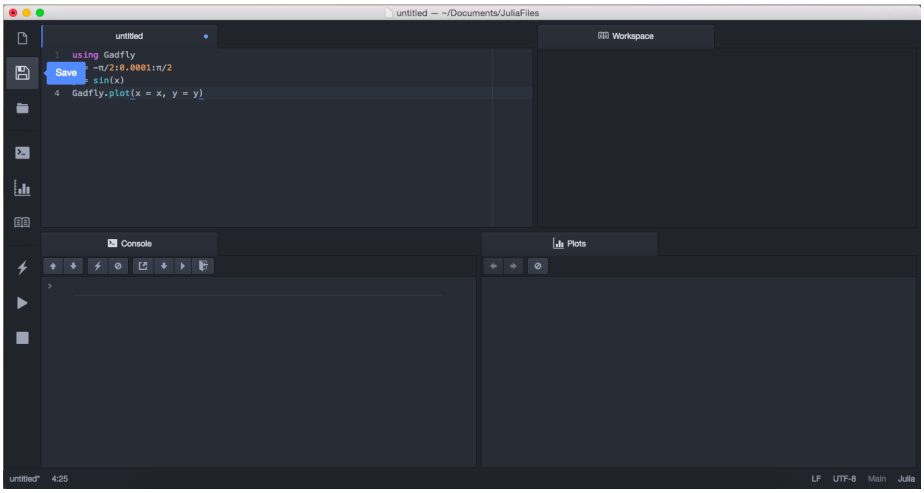
The Julia Toolbar contains buttons that represent the following functions (listed in vertical order from top to bottom when toolbar is in its default left location):

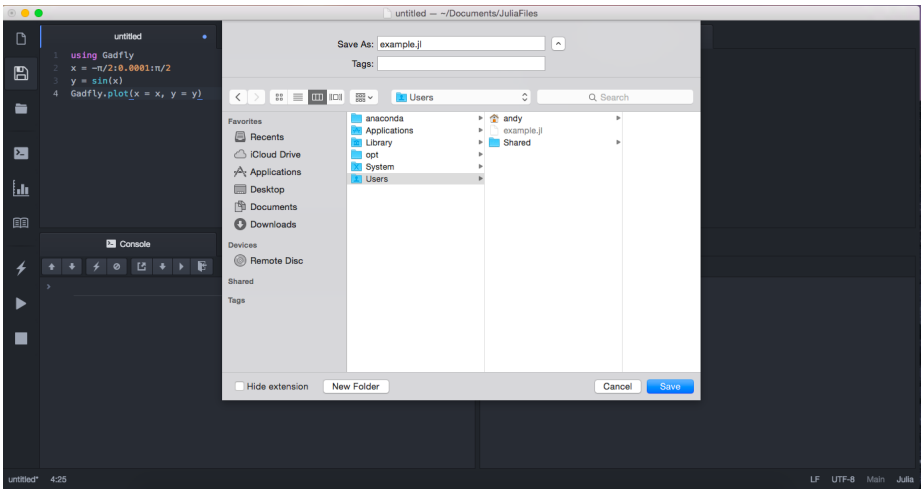
- New Julia File
- Save
- Open File
- Open Console
- Show Plots
- Run Block
- Run File
- Stop Julia

The "New Julia File" button will create a new Editor Tab aassociated with a new Julia source file.

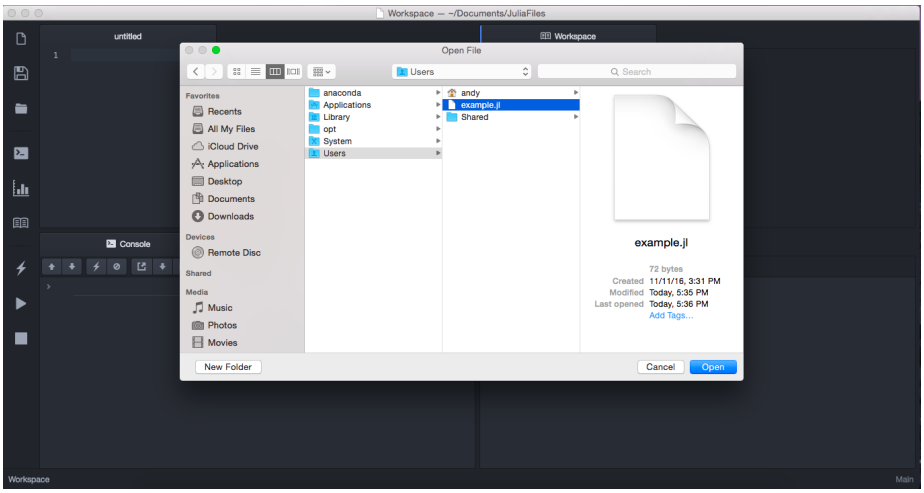


The "Save" button will open a pop-up window allowing the user to save the currently active Julia source file to a user defined location.

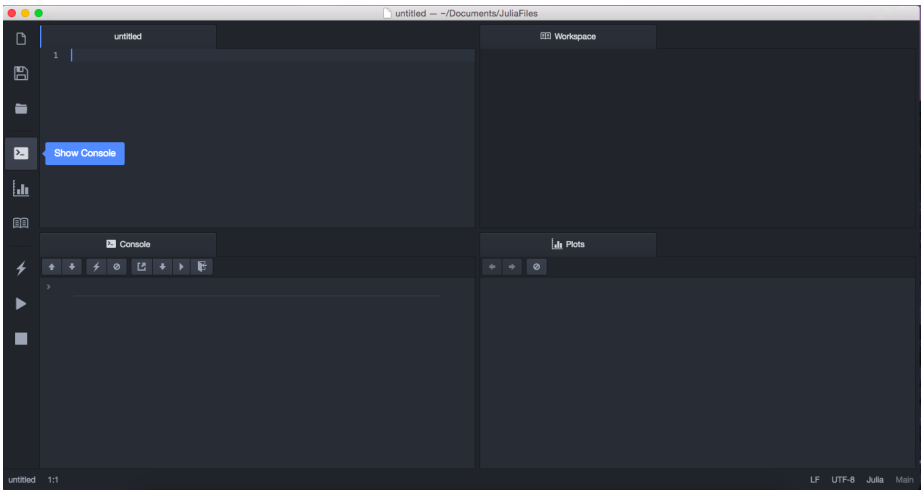


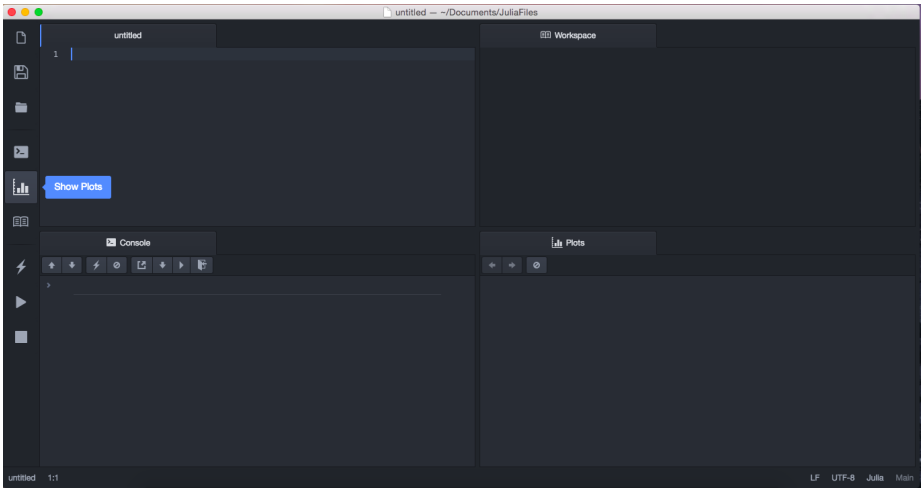


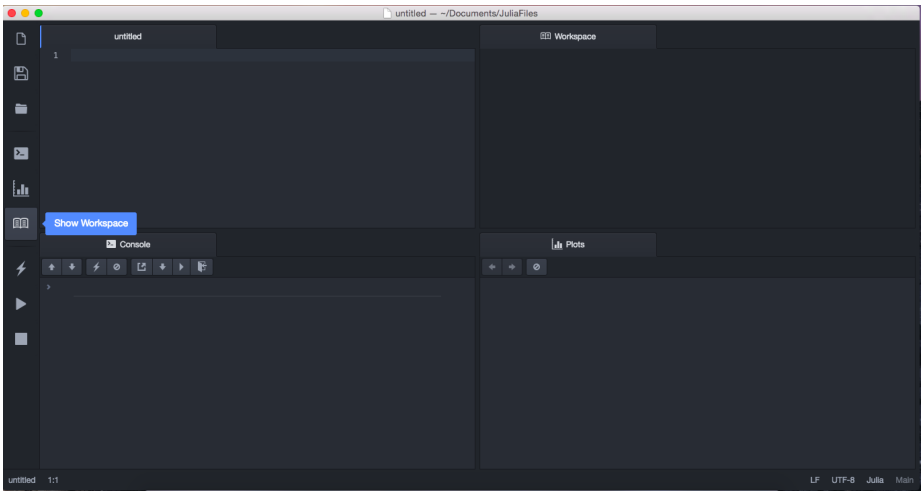
Similarly, the "Open File…" button allows the user to select a source file that they would like to open in the current editor pane.



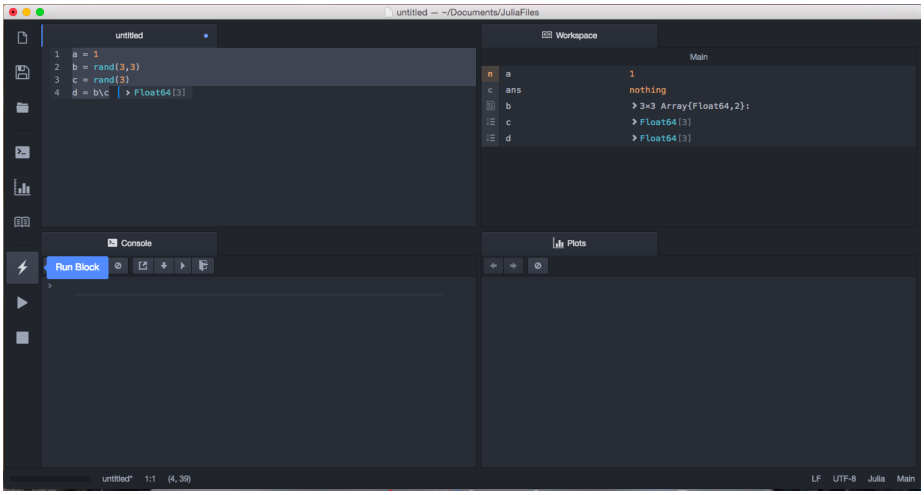
If any of the Console tab, the Plots tab, or the Workspace tab has been closed, then the "Show Console", "Show Plots", or "Show Workspace" buttons can be used to re-open those particular tabs.


untitled — ~/Documents/JuliaFiles
untitled
Workspace
Show Console
Console
Plots
untitled 1:1
LF UTF-8 Julia Main

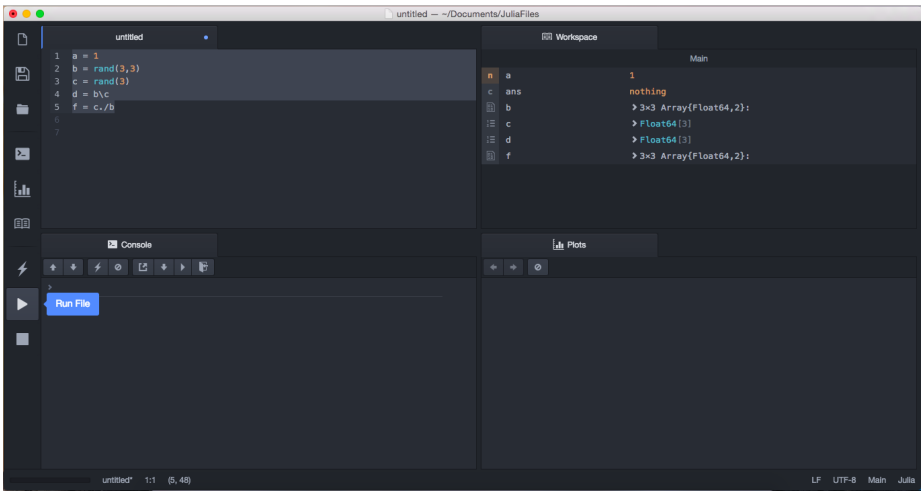

untitled — ~/Documents/JuliaFiles
untitled
Workspace
Show Plots
Console
Plots
untitled 1:1
LF UTF-8 Julia Main



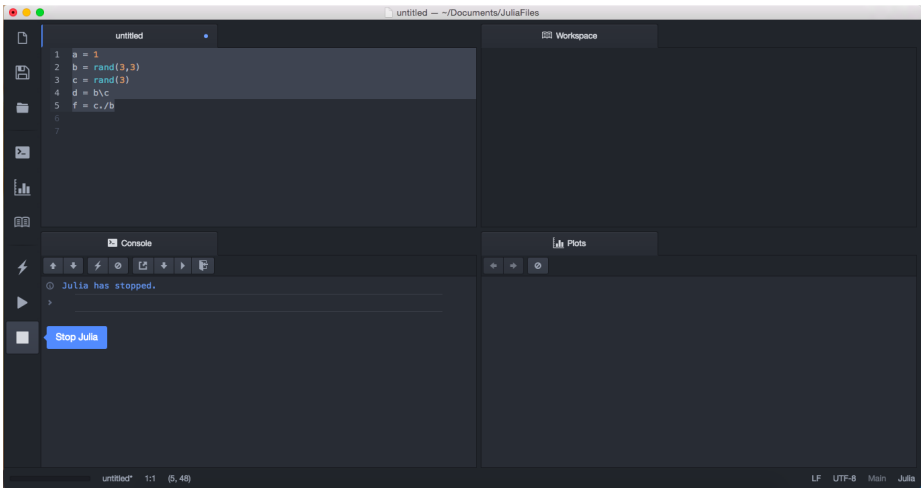
The "Run Block" button will execute a series of highlighted statements from within the editor. This command can also be accessed via the keyboard shortcut Cmd-Enter.



The "Run File" button will execute the entirety of the source file that is currently active in the editor window. If there is no active Julia kernel when this button is pressed, then a new Julia kernel will be launched. Files can also be run via Cmd-Shift-Enter.



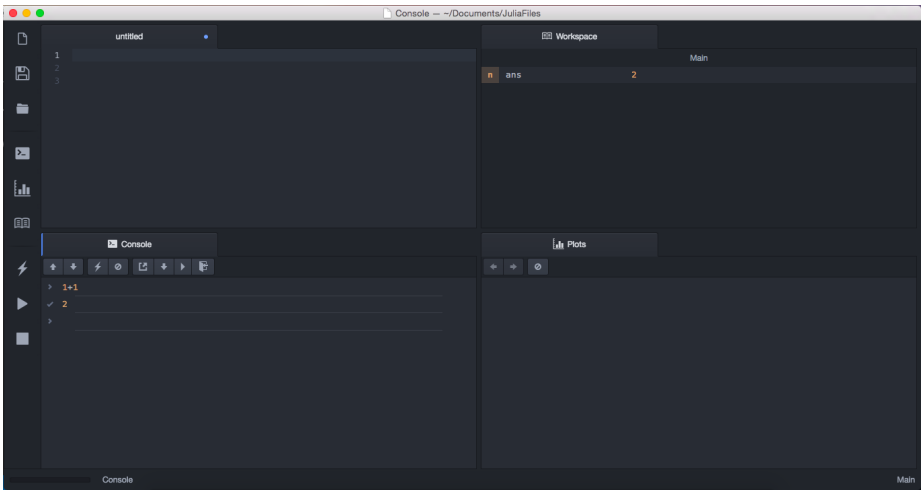
The “Stop Julia” button will stop the currently active Julia kernel.



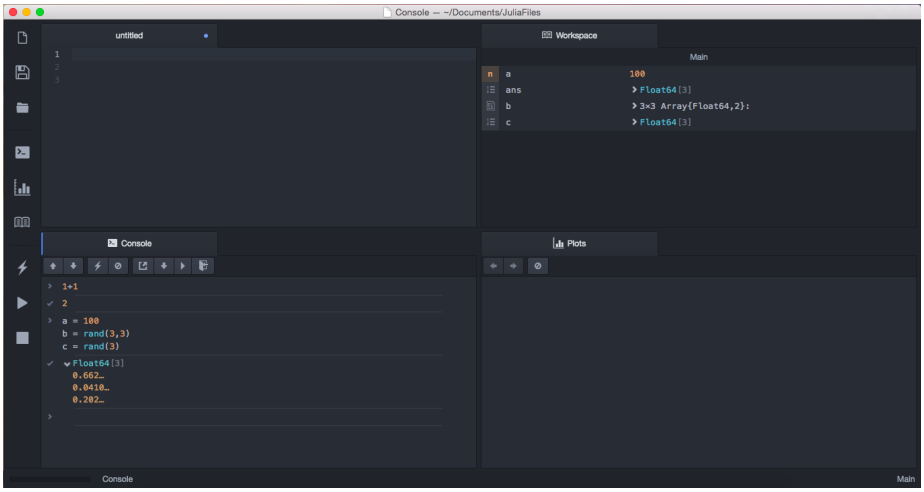
8.6 5.2.2 Using the Console Tab

The Console tab provides access to a command prompt that allows for immediate execution of Julia statements. The Console tab also includes its own integrated toolbar for controling the state of the Console prompt as well as debugging Julia code.

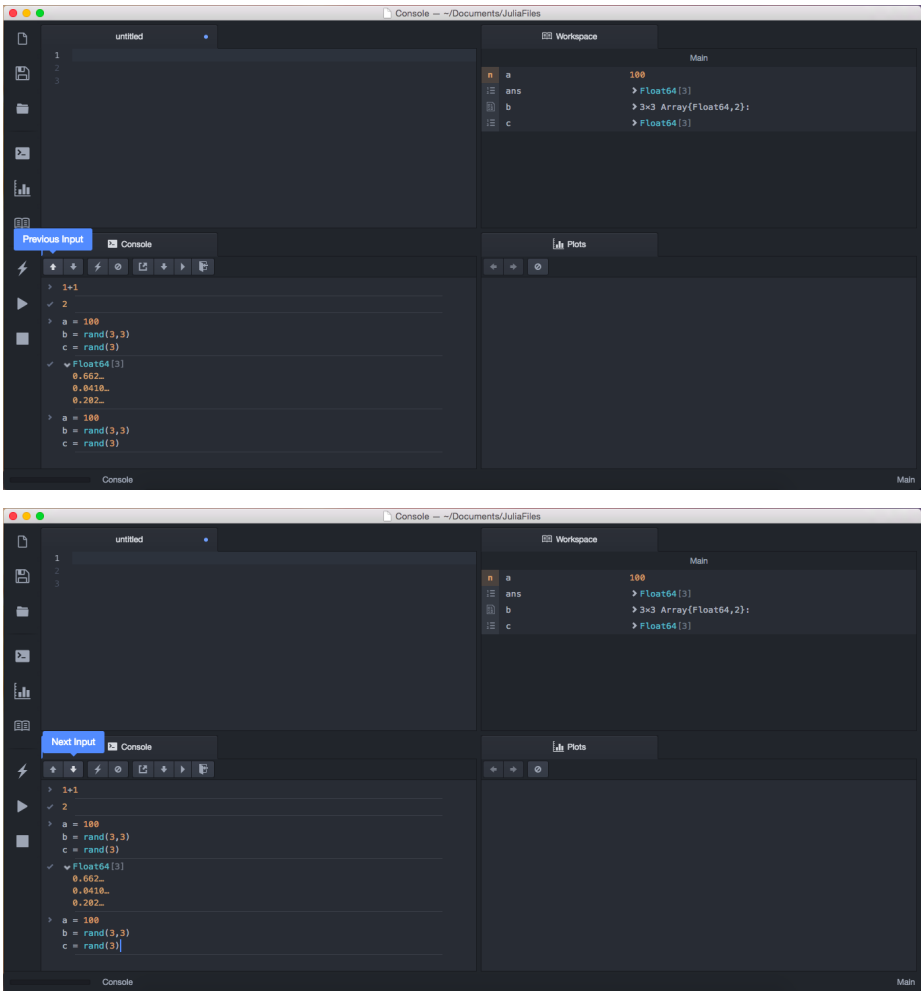
Executing a single Julia statement within the Console tab requires either pressing Enter upon completing the entry of the command, or pushing the Run button in the Console toolbar.



To enter a multiline command within the Console tab, press Shift+Enter upon the completion of a line in the Console. Subsequently pressing the Enter key or pushing the Run button will execute all of the commands in a multiline statement.

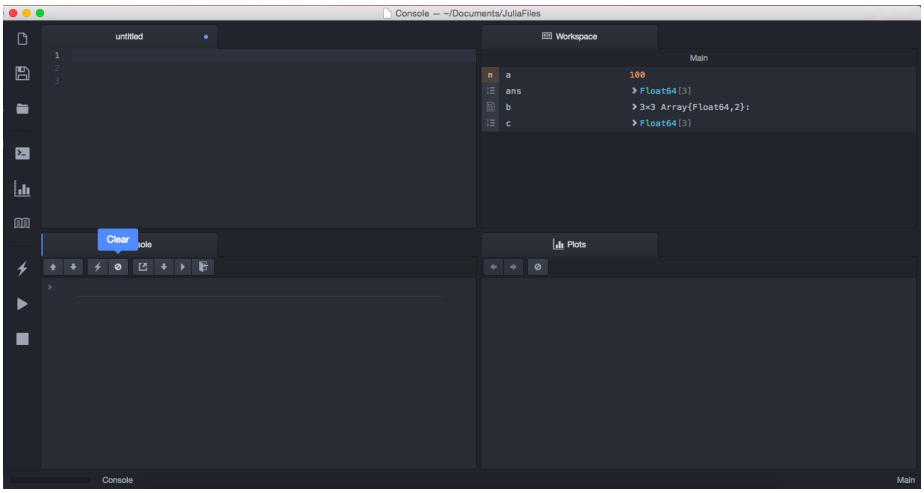


Scrolling through previously entered commands in the Console tab can be accomplished via either the up and down keys on your keyboard, or via the Previous Input and Next Input buttons in the Console toolbar.



Like in the main Julia toolbar, the Run button can be used for executing single line or multiline commands.

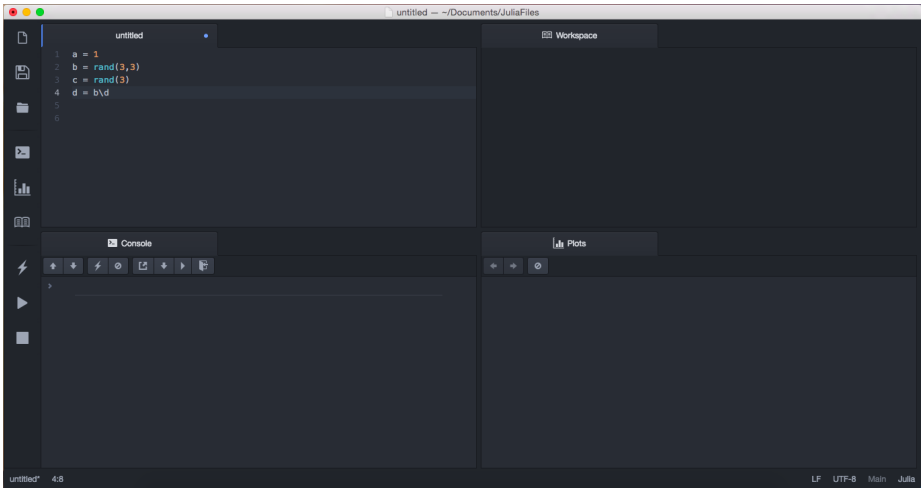
The currently displayed text within the Console tab can also be cleared via the Clear button.



The remaining buttons in the Console toolbar are used for the debugging of Julia code and are discussed in the section Debugging Using the Console and Editor Tabs below.

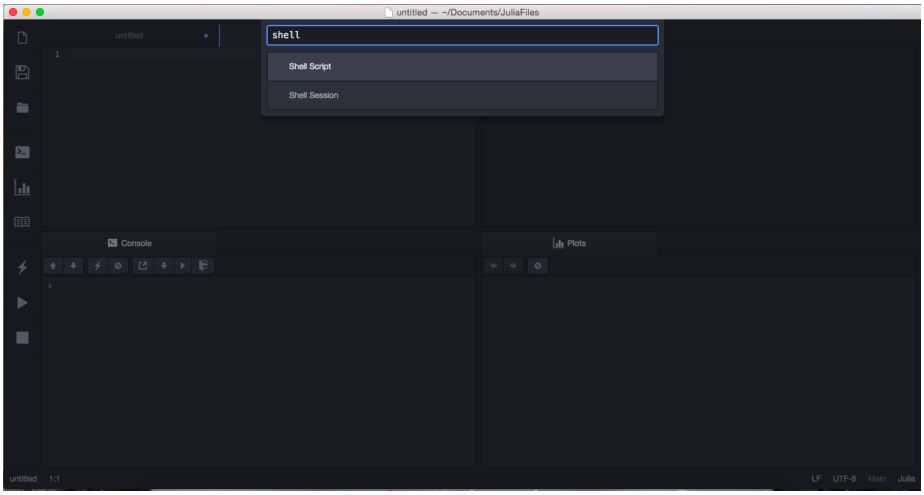
8.7 5.2.3 Using the Editor Tab

The Editor tab can be used for developing any Julia scripts, modules, or packages that can be saved as *.jl files.

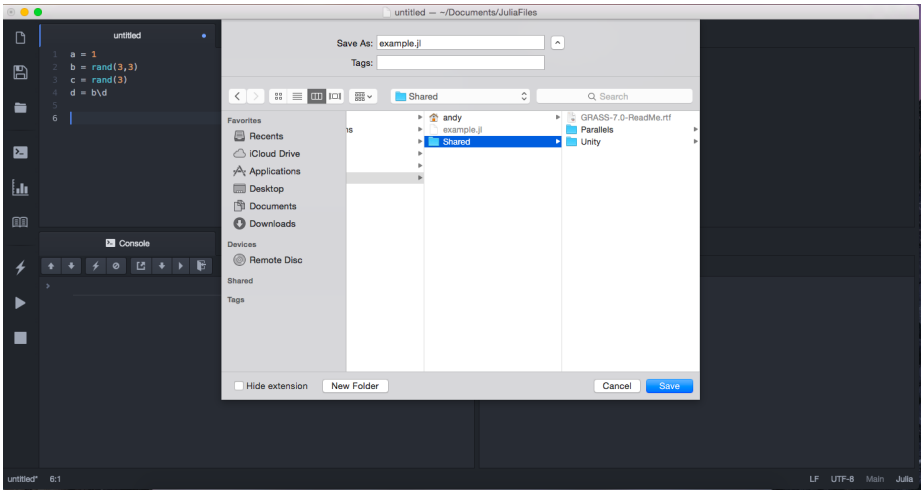


While opening a new source file defaults to associating that file with Julia source code (providing associated syntax highlighting), as Juno is built on Atom, a given source file can also be associated with any language supported by the Atom editor. The language mode for the current editor tab can be modified

by clicking on the “Julia” entry in the bottom right hand corner of the editor window.

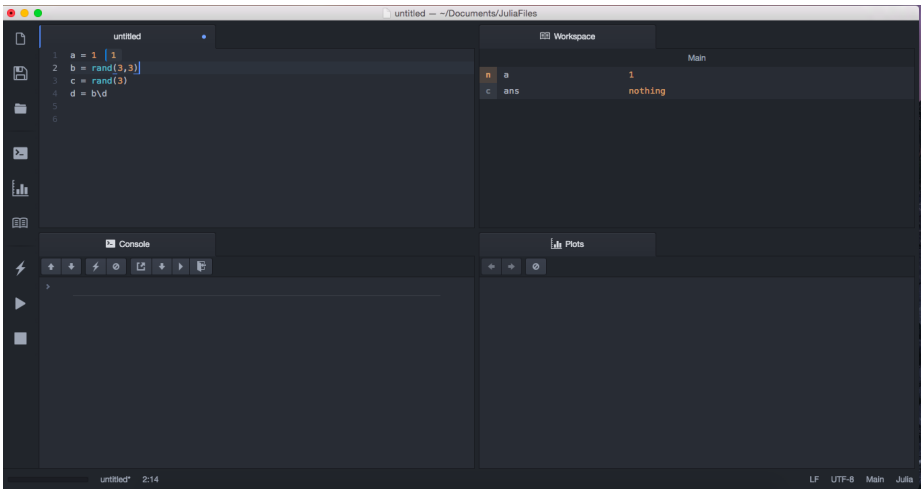


Upon editing the current source file, any edits can be saved by either pressing the Cmd+S keystroke combination, pressing the Save button in the main Julia toolbar, or selecting “File -> Save” from the Atom menu bar.

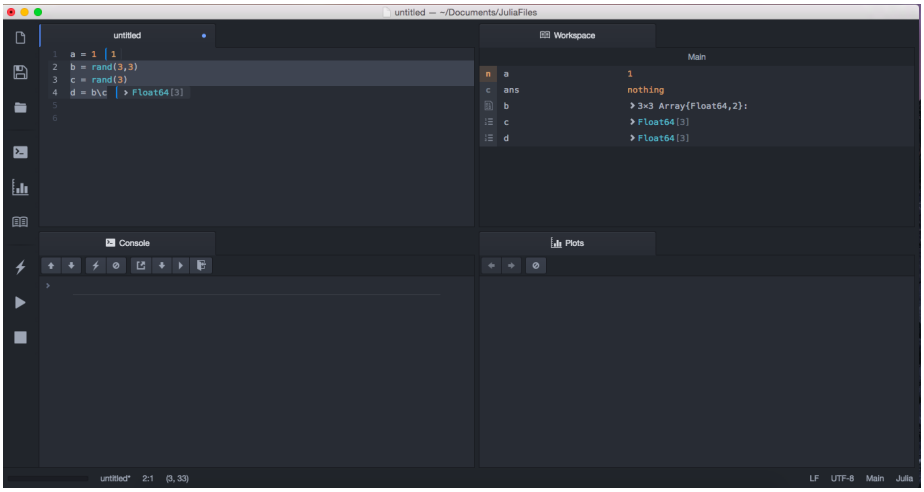


As stated previously, in addition to being used just for editing code, Julia source code can also be executed directly from within the Editor tab.

A single statement can be executed within the Editor by placing the cursor on the desired source line and then pressing Shift+Enter or pressing the “Run Block” button.



Multiple statements can also be executed at once in the Editor by highlighting multiple lines before pressing Shift+Enter or pressing the "Run Block" Button.

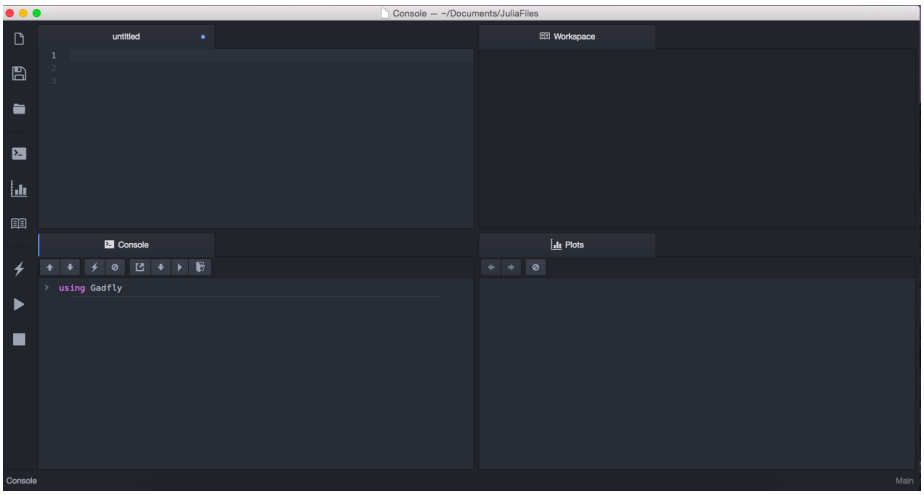


8.8 5.2.4 Using the Plots Tab

When using the Gadfly.jl visualization package, plots can be displayed within Juno's Plots tab.

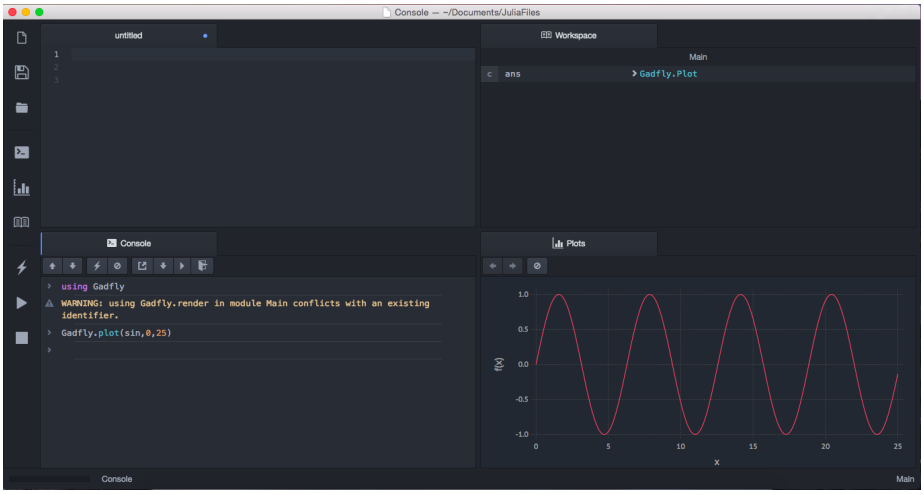
To create a plot using Gadfly, one first needs to load the Gadfly package as follows:

```
> using Gadfly
```



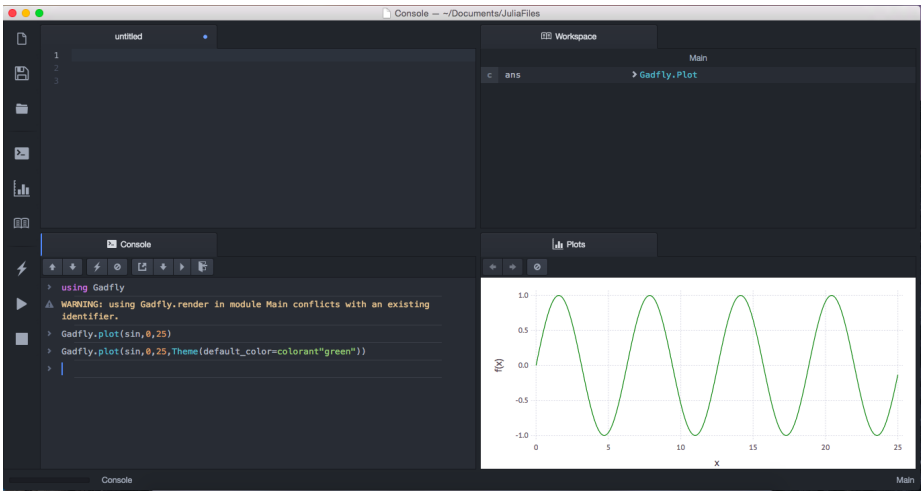
One can create a simple plot of a sine curve as follows:

```
Gadfly.plot(sin,0,25)
```



To alter the color of the line, the `Gadfly.plot` command can be altered in the following manner:

```
Gadfly.plot(sin,0,25,Theme(default_color=colorant"green"))
```



For visualizations produced via Gadfly.jl, executing its plot command creates a static image of the plot, so currently the pan and zoom functionality that is present when using Gadfly to plot in the browser is not available.

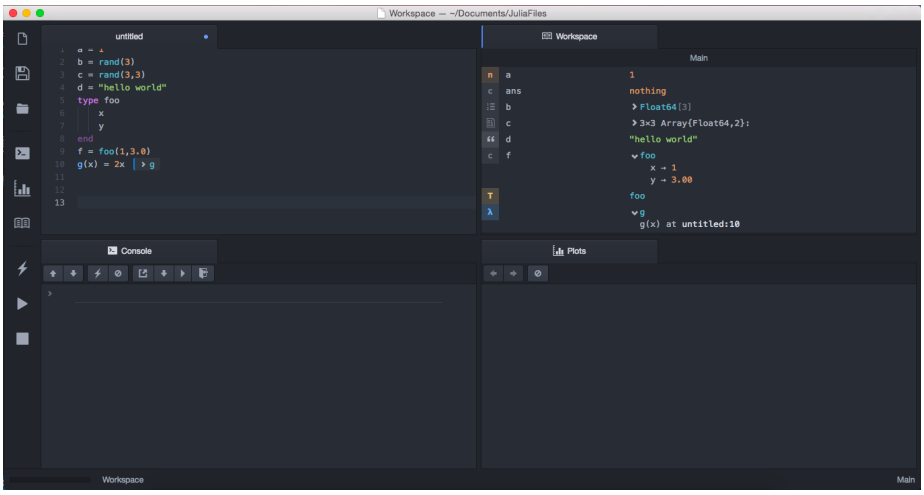
See the Gadfly documentation for its full list of options for customizing a plot.

8.9 5.2.5 Using the Workspace Tab

The Workspace tab displays a summary view of all variables, types, and functions that are defined within the current scope of the current Julia session. The entries within the Workspace tab consist of 3 columns:

- One of the following icons for the kind of entry displayed in the list of Workspace entries
- n
- c
- T
- "
- vector
- Array
- The name of any assigned variables
- The value of assigned variables or name of any defined function or type

For any array variables, instances of a user defined type, or function definitions, the contents of those objects can be inspected in more detail by clicking on the > symbol adjacent to an entry in the value column.

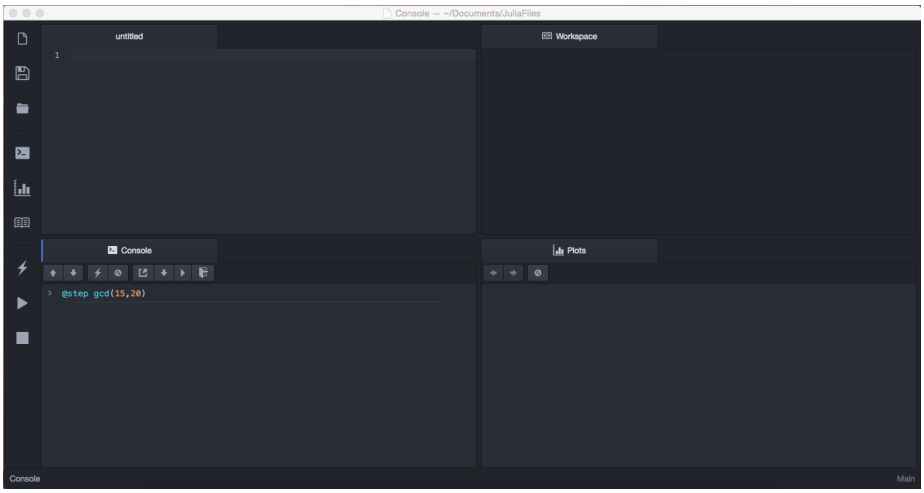


8.10 5.2.6 Debugging Using the Console and Editor Tabs

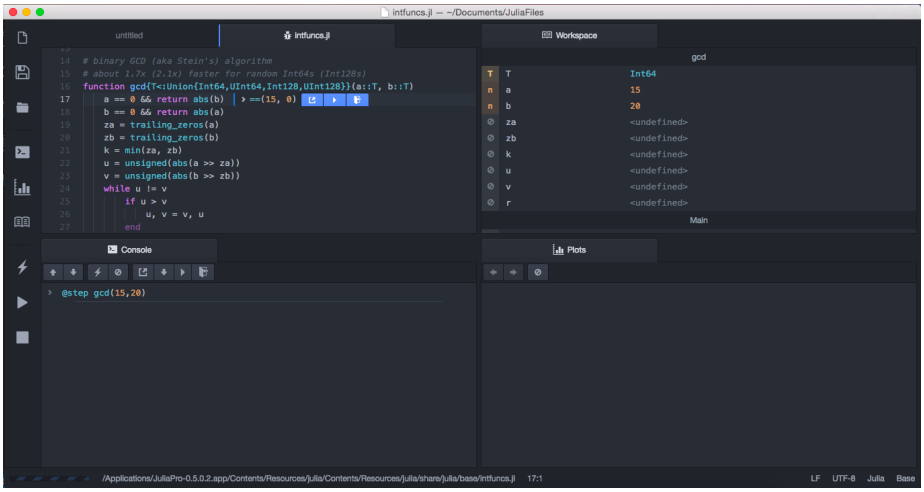
The final four buttons in the Console Toolbar allow for use of functionality from Julia's Gallium debugger from within Juno. These four buttons include:

- Debug: Step to Next Line
- Debug: Finish Function
- Debug: Step into Function
- Debug: Step to Next Expression

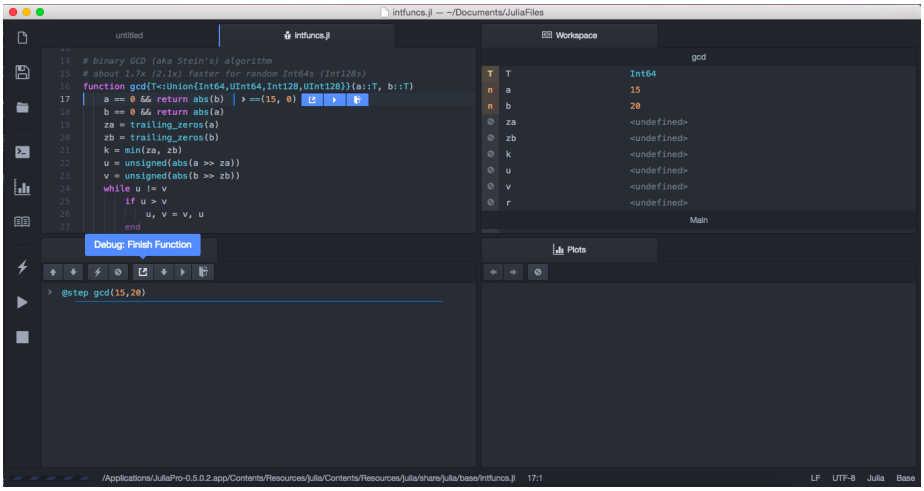
Use of debugging functionality from within Juno is associated with use of the `@step` macro.



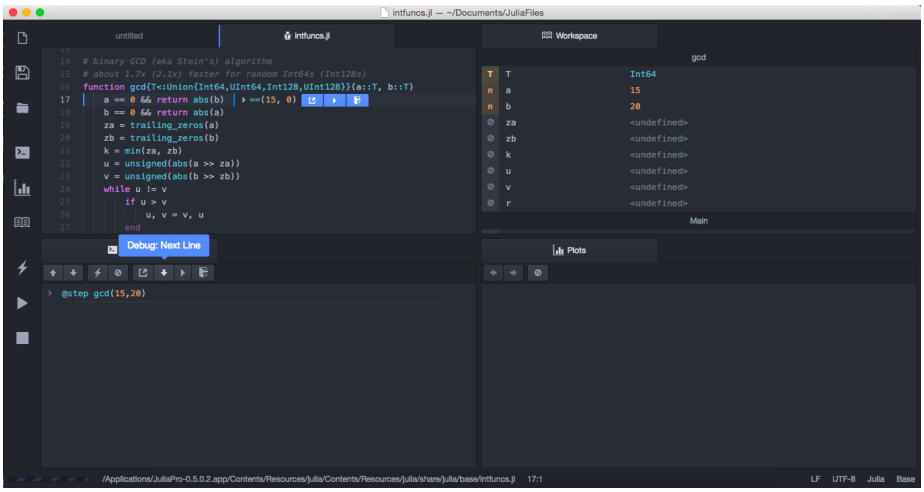
Upon executing a statement preceded by the `@step` macro, the source file containing the relevant method will open within an editor pane, and a small pop-up toolbar is presented at the line where a Gallium breakpoint has been hit. The Workspace window will also display variables in the scope of the method where the debugger is currently stopped.



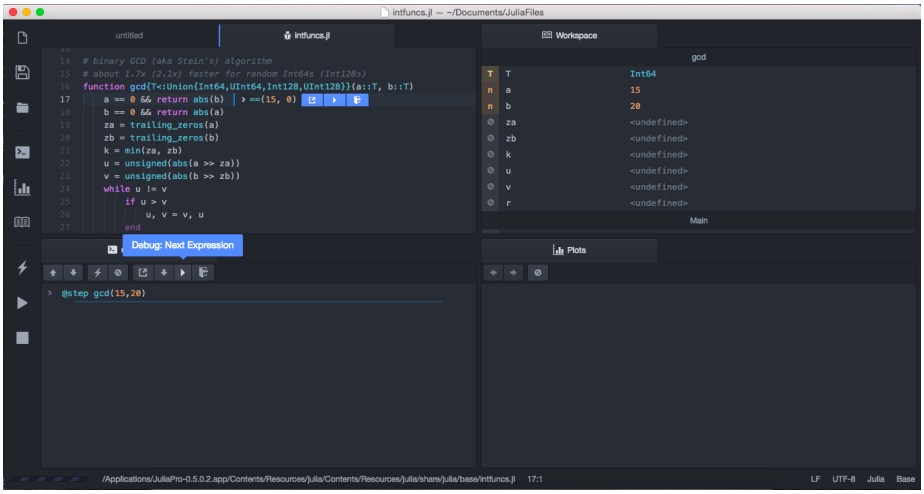
The "Debug: Step to Next Line" button executes the active line in the current source file and stops at the subsequent line.



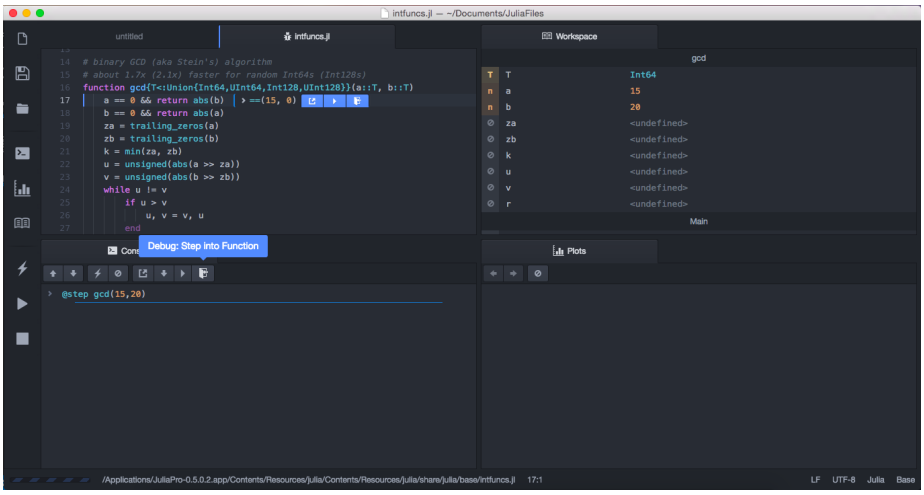
The "Debug: Finish Function" button executes all statements included in the function where the currently active debugger stop is located and returns to the scope from which the current function was called.



The "Debug: Step Into Function" button advances execution of the program to the first line of the function where the debugger is currently stopped and sets the active state of the debugger to be located in that function's source file.

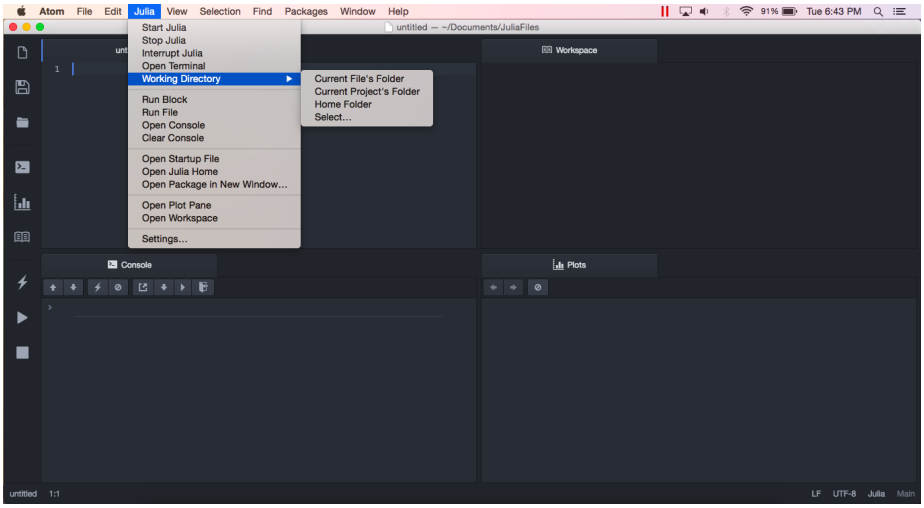


The "Debug: Step to Next Expression" button advances execution of the program to the next expression in the active line of the current source file. Use of this button allows one to proceed from one subexpression to another on a single line, and progressively execute and inspect nested expressions on a single line.



8.11 5.3 Julia Menu

The Julia Menu entry in the Juno menu bar includes a variety of functionality for configuring your Juno session for use with Julia.



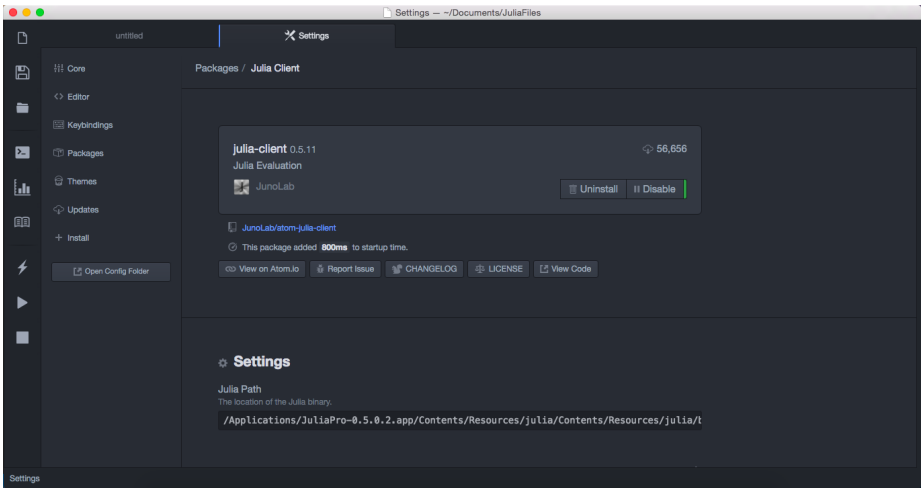
The Julia menu includes the following entries:

- Start Julia – starts a new Julia kernel if one is not currently attached to the current Atom session
- Stop Julia – stops the currently attached Julia kernel
- Interrupt Julia – interrupts the currently attached Julia kernel
- Open Terminal – opens a separate Julia terminal window with a new Julia kernel

- Working Directory – allows for changing the current working directory to either the Current File's Folder, the Current Project's Folder, the user's Home Folder, or an arbitrary location
- Run Block – executes a currently highlighted block of code within the active source file
- Run File – executes the entirety of the currently active source file
- Open Console – opens the Julia console window if not already open and launches a new Julia kernel
- Clear Console – clears the printed contents of the Julia console pane
- Open Startup File – opens the current user's .juliarc.jl file allowing for the addition of commands that are automatically executed on startup of Julia.
- Open Julia Home – opens the Julia Home directory associated with
- Open Package in New Window... - opens the directory containing the package associated with the currently active file
- Open Plot Pane – opens the Julia plot pane if not already opened
- Open Workspace – opens a Workspace pane used for the display of variables present in the scope of the currently active Julia kernel.
- Settings... - opens a copy of the Julia settings pane for configuring the julia-client package for Juno

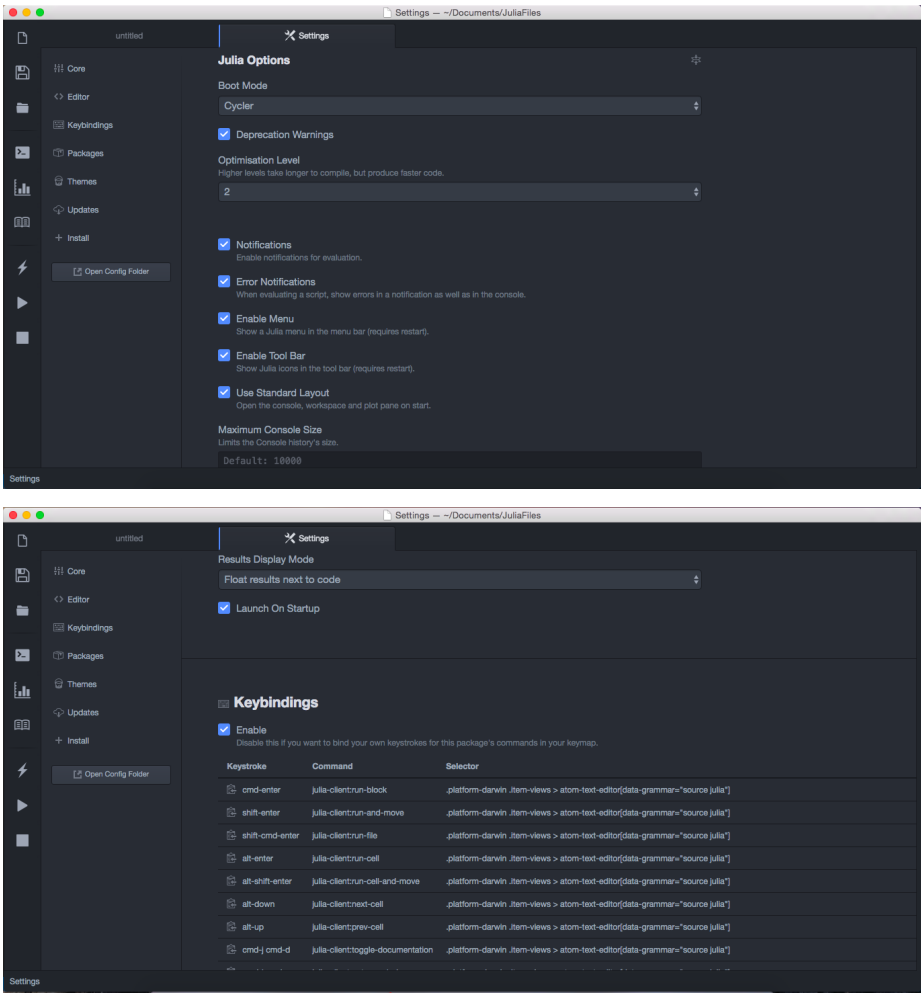
8.12 5.3.1 Julia Settings Tab

The Julia settings tab available from the Julia menu allows for updating various settings available as part of the julia-client Atom package.



Available Julia Settings include:

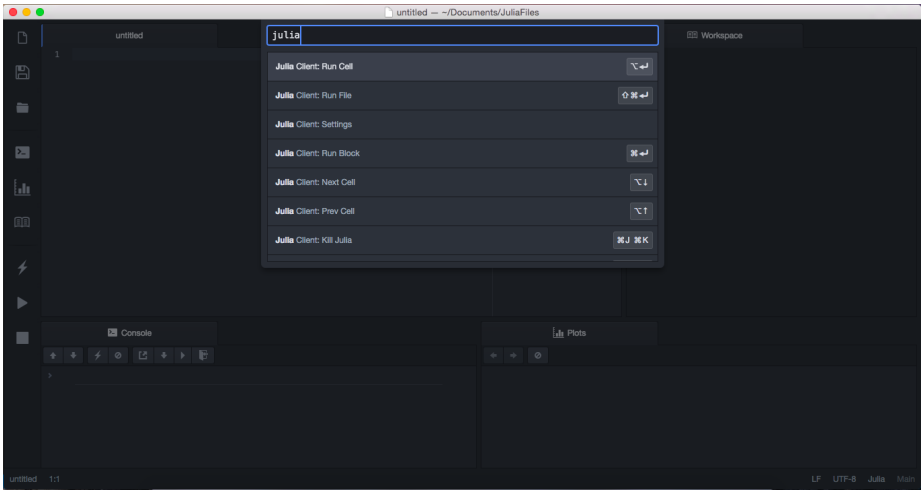
- Julia Path – the path to the location of the Julia binary. It is not recommended to update this setting manually as part of a JuliaPro installation.



Available Julia Options include: * Boot Mode – options include Basic, Cycler and Server. It is not recommended to update this setting for a JuliaPro installation. * Deprecation warnings – display deprecation warnings * Optimization Level – the level of optimizations used in compiling Julia code. Higher levels take longer to compile but produce faster code. Available levels include values x through y * Notifications – Enable notifications for evaluation * Error Notifications – When evaluating a Julia script, show errors in a notification windows as well as in the console. * Enable Menu – Show the Julia menu in the Juno menu bar. Activating or deactivating this setting requires restarting Juno. * Enable Toolbar – Show the Julia icons in the toolbar. Activating or deactivating this setting requires restarting Juno. * Maximum Console Size – A maximum limit on the size of the Julia history in the Console pane * Launch on Startup – launches a Julia kernel on startup of Juno. * Keybindings - A list of current keybindings for the Juno package within Atom.

8.13 5.4 Julia Commands in the Command Palette

The Command Palette provides access to all Julia-related commands available for use from within the Atom IDE. The Command Palette can be opened either by executing the Cmd+Shift+P or selecting "Edit -> View -> Toggle Command Palette" from the Juno menu. Available Julia commands can be seen by typing "julia" into the Command Palette. Keyboard shortcuts are also listed on the right of the command description.



The commands available from the Command Palette include the following:

- Julia Client: Settings
- Julia Client: Run File
- Julia Client: Run Block
- Julia Client: Kill Julia (Cmd+J Cmd+K)
- Julia Client: Goto Symbol
- Julia Client: Open a Repl (Cmd+J Cmd+R)
- Julia Client: Start Julia (Cmd+J Cmd+S)
- Julia Client: Open Console (Cmd+J Cmd+O)
- Julia Client: Run and Move
- Julia Client: Select Block
- Julia Client: Send To Stdin
- Julia Client: Clear Console (Cmd+J Cmd+C)
- Julia Debug: Finish Function
- Julia Client: Open Workspace
- Julia Client: Open Plot Pane (Cmd+J Cmd+P)
- Julia Client: Reset Workspace
- Julia Client: Interrupt Julia
- Julia Debug: Step to Next Line

- Julia Debug: Toggle Breakpoint
- Julia: Open Julia Home
- Julia Debug: Step Into Function
- Julia Client: Reset Julia Server
- Julia: Open Startup File
- Juila Client: Work In File Folder
- Julia Client: Work In Home Folder
- Julia Client: Toggle Documentation
- Julia Client: Connect to Local Port
- Julia Client: Select Working Folder
- Julia Debug: Step to Next Expression
- Julia Client: Work in Project Folder
- Julia: Open Package in New Window
- Language Julia: Toggle Docstrings
- Language Julia: Toggle All Docstrings