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 additional packages, including JuliaPro's Juno IDE, included Jupyter notebook functionality, as well as the JuliaInXL package.

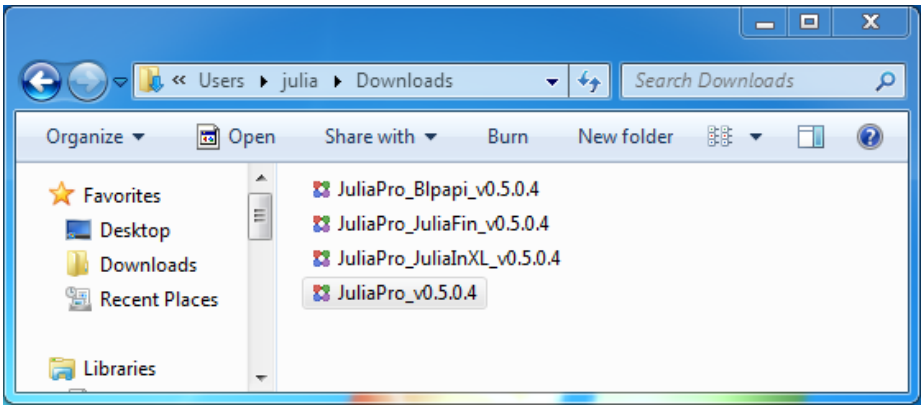
5 2. Prerequisites

- An appropriate version of Microsoft® Windows®
- Windows 7 SP1, Windows 8, Windows 8.1, Windows 10
- Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2, Windows Server 2016
- Julia 0.5.x (bundled with the base JuliaPro installer)
- 5 GB of disk space
- .NET 4.0 (required for use of JuliaInXL)
- Bundled with the JuliaInXL installer when executed with Administrator privileges
- Microsoft Excel® 2010, 2013, or 2016 (required for use of JuliaInXL)

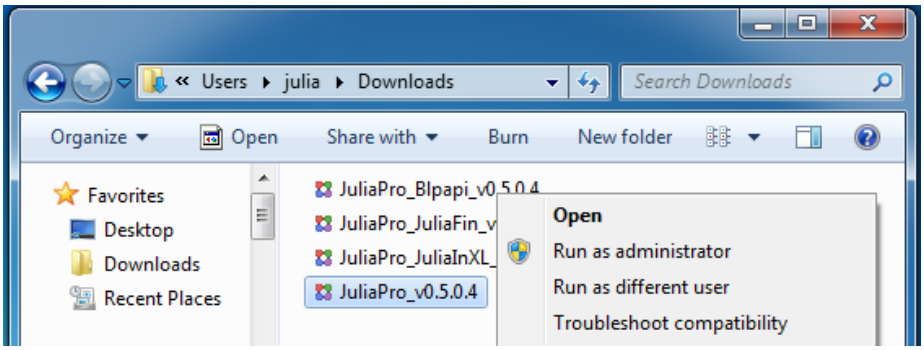
6 3. Installation

6.1 3.1 Installing JuliaPro

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

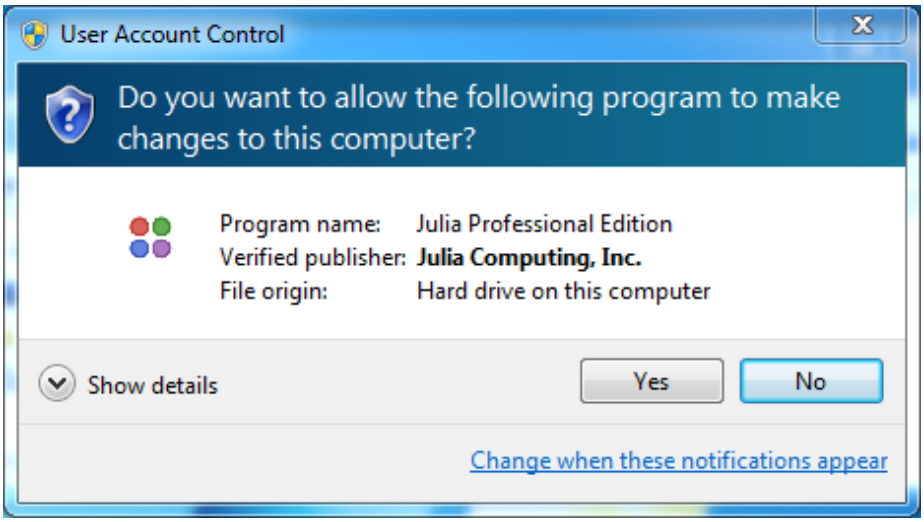


To execute the installer with Administrative privileges, right click on the JuliaPro_v0.5.0.4.exe and select "Run As Administrator". To execute the installer as the current user, just double-click on the executable.

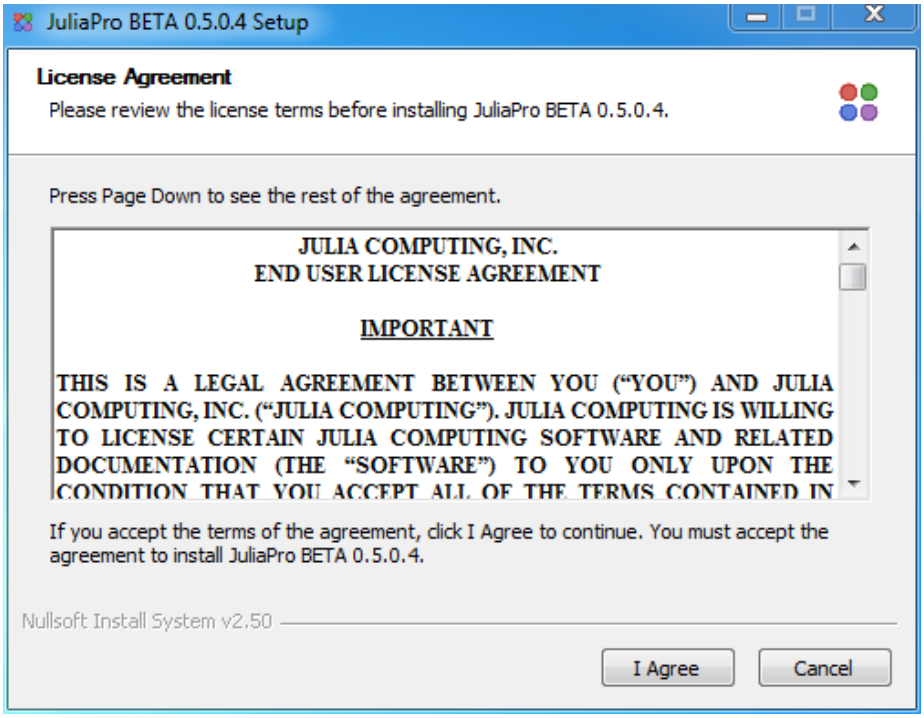


Upon launching the executable, you might be presented with a User Account Control permissions window.

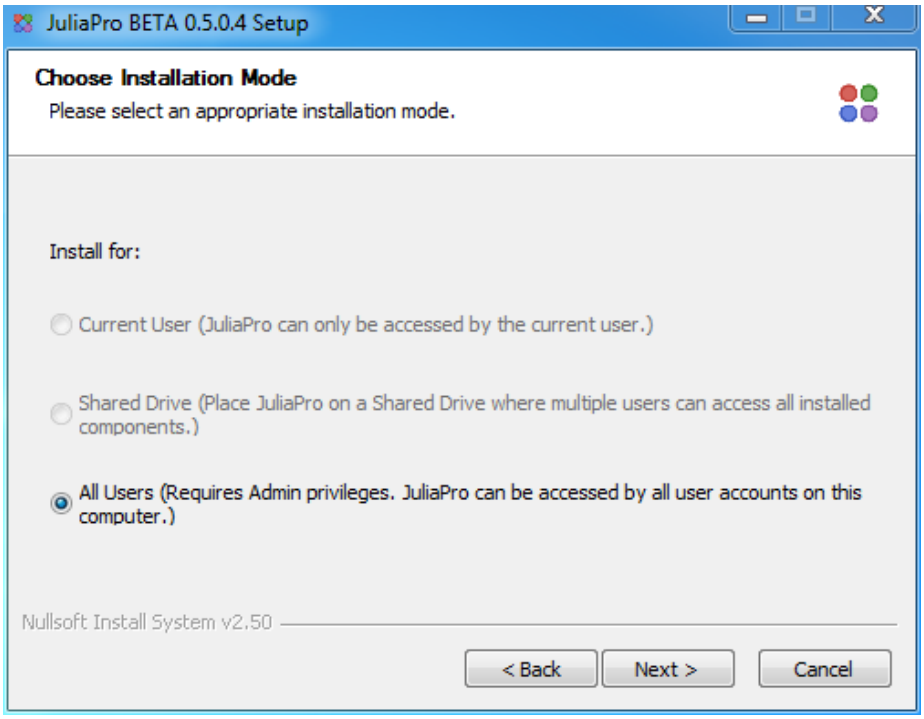
If the installer is being executed with Administrative privileges, then click "Yes" to proceed with the installation.



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

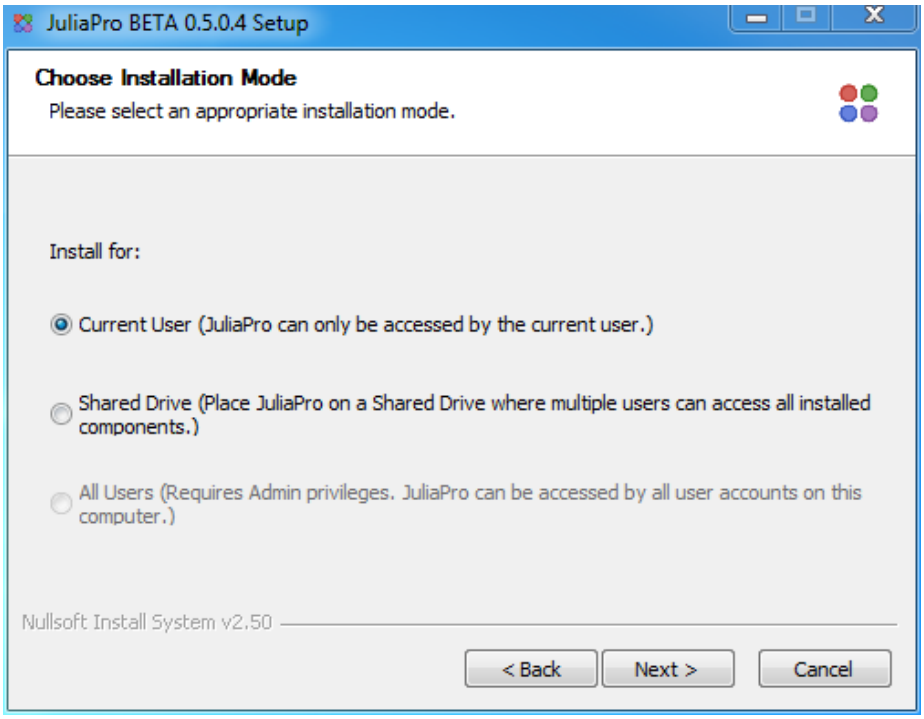


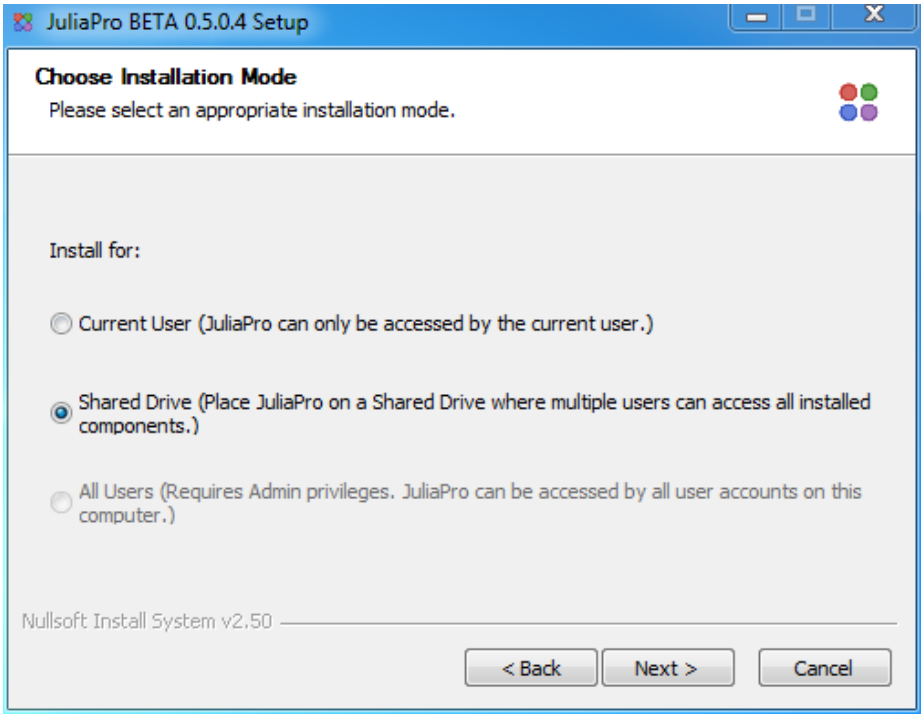
If the installer is being executed with Administrative privileges, then you will be presented with the following installation option:



- **All Users:** This installation option requires the current user to have Administrator privileges and will default to installing JuliaPro to a location available to all system users. The default location is a folder created at the location of the current %HOMEDRIVE% environment variable. Selecting this option also allows for a user with Administrative privileges to install Julia Professional to a shared network drive location.

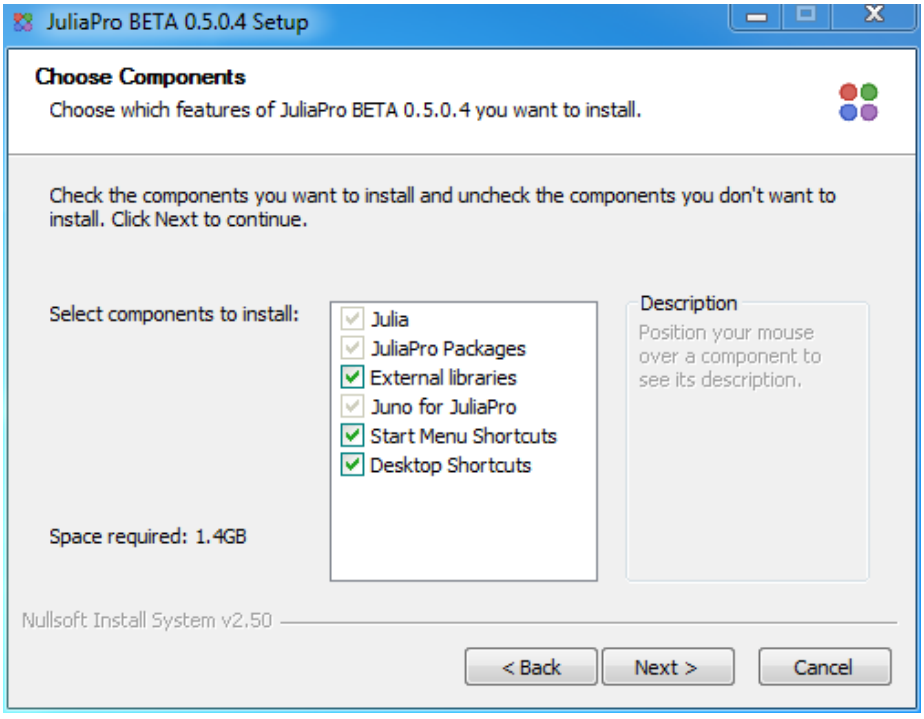
If the installer is being executed without Administrative privileges, then you will be presented with the following two active installation options:


JuliaPro BETA 0.5.0.4 Setup
Choose Installation Mode
Please select an appropriate installation mode.
Install for:
Current User (JuliaPro can only be accessed by the current user.)
Shared Drive (Place JuliaPro on a Shared Drive where multiple users can access all installed components.)
All Users (Requires Admin privileges. JuliaPro can be accessed by all user accounts on this computer.)
Nullsoft Install System v2.50
< Back
Next >
Cancel

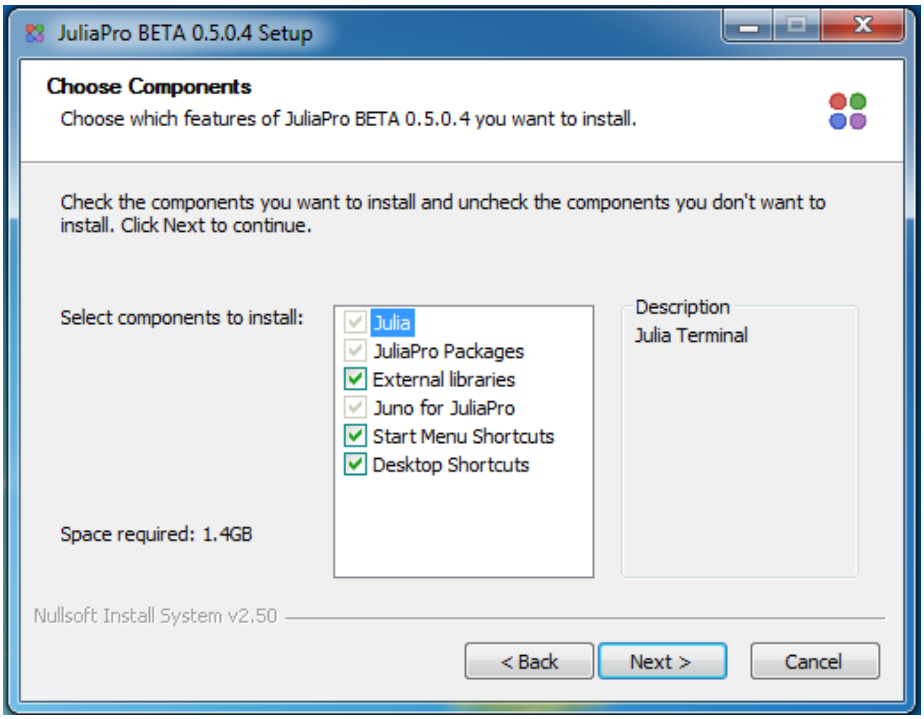

JuliaPro BETA 0.5.0.4 Setup
Choose Installation Mode
Please select an appropriate installation mode.
Install for:
Current User (JuliaPro can only be accessed by the current user.)
Shared Drive (Place JuliaPro on a Shared Drive where multiple users can access all installed components.)
All Users (Requires Admin privileges. JuliaPro can be accessed by all user accounts on this computer.)
Nullsoft Install System v2.50
< Back
Next >
Cancel

- **Current User (No Admin Privileges Required):** With this installation option, the selected directory into which JuliaPro is installed must be a directory for which the current user has read/write privileges. The default directory chosen is within the "AppData" directory of the current user's account.
- **Shared Drive (No Admin Privileges Required):** With this installation option, the selected directory into which JuliaPro is installed can be a shared network drive location for which the current user has read/write privileges. While the default directory chosen is within the "AppData" directory of the current user's account, this option is useful when the current user intends to install Julia Professional to a shared location, but does not have administrative privileges.

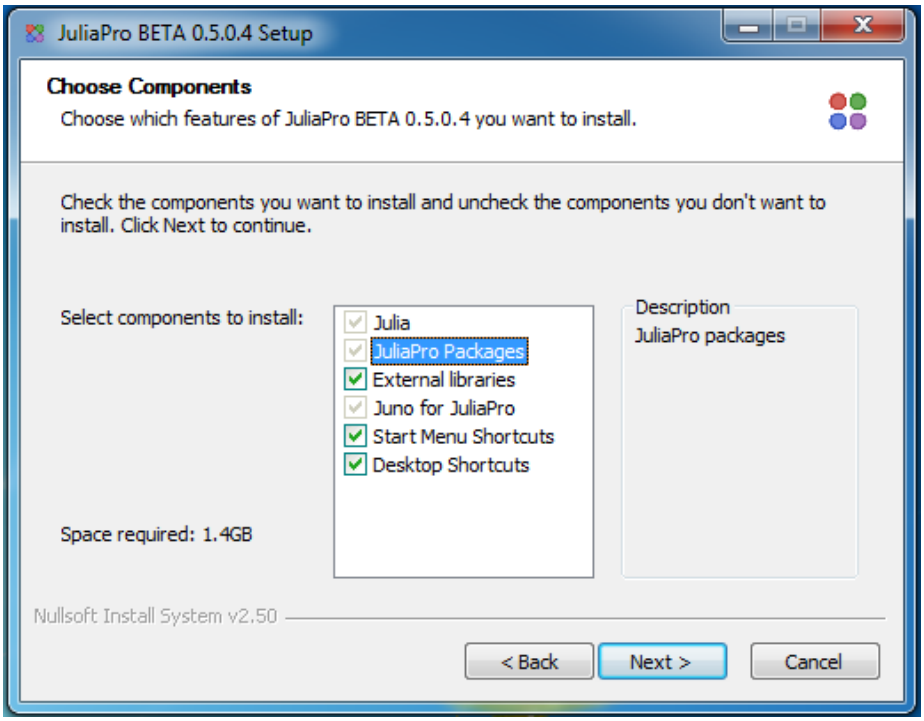
After selecting the type of installation to perform, you will be presented with options on which components of the JuliaPro distribution you would like to install.



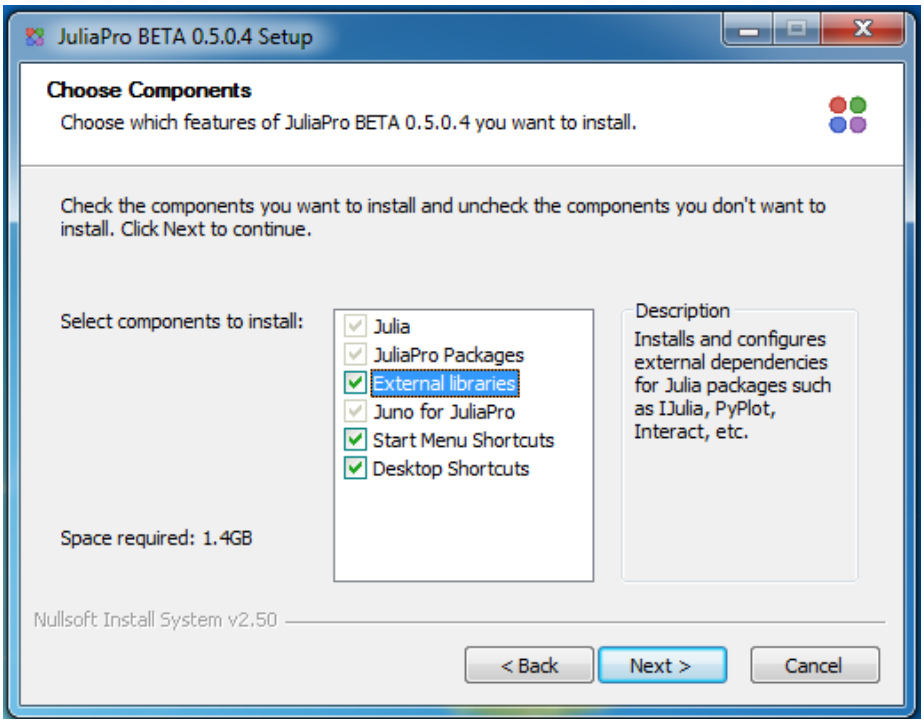
The first listed component is the Julia compiler, runtime and terminal application. This component is required for installation.



The next listed component is the set of all Julia packages to be installed as part of JuliaPro. This component is also required for installation.

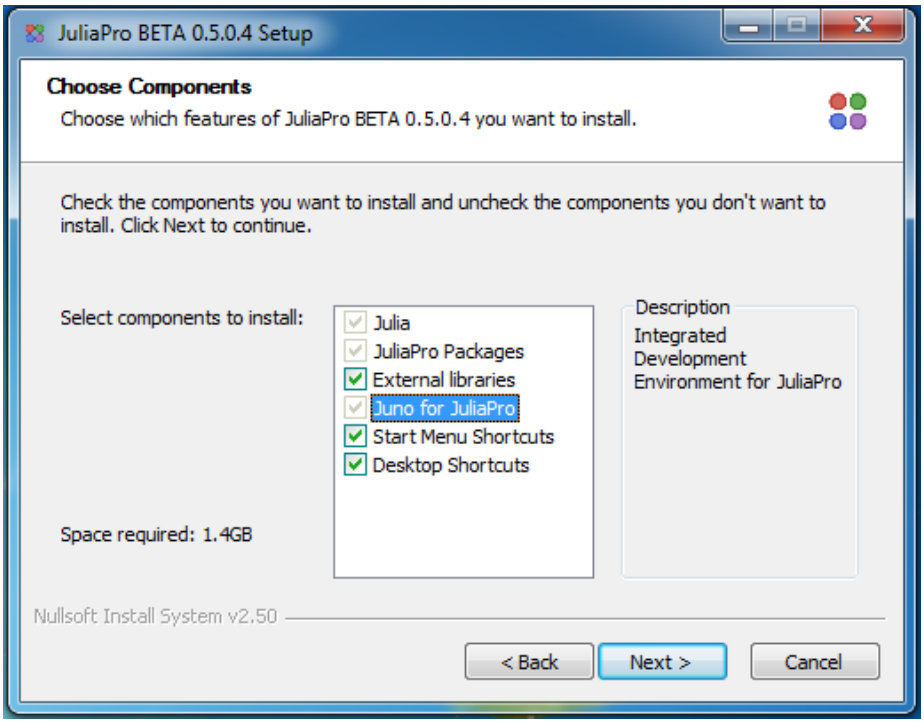


The third listed compoent is a set of external Python language dependencies necessary for use of the IJulia, PyCall, PyPlot, and Interact packages. If selected, this component will include a private installation of Python, Matplotlib, Jupyter, and various necessary Python dependencies for use of those packages.

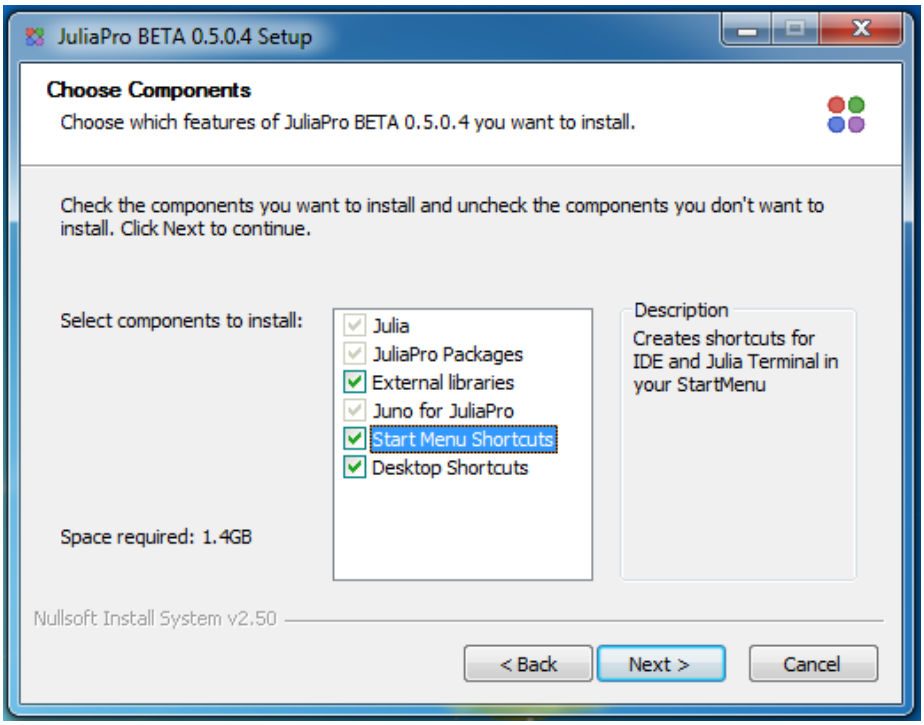


It is possible to configure the PyCall and IJulia packages to use and existing installation of Python and appropriate packages that might already be installed on your system, in which case the installation of the External Dependencies component is optional. Later sections will describe how to configure the PyCall and IJulia packages distributed with JuliaPro to make use of an existing Python installation on your system.

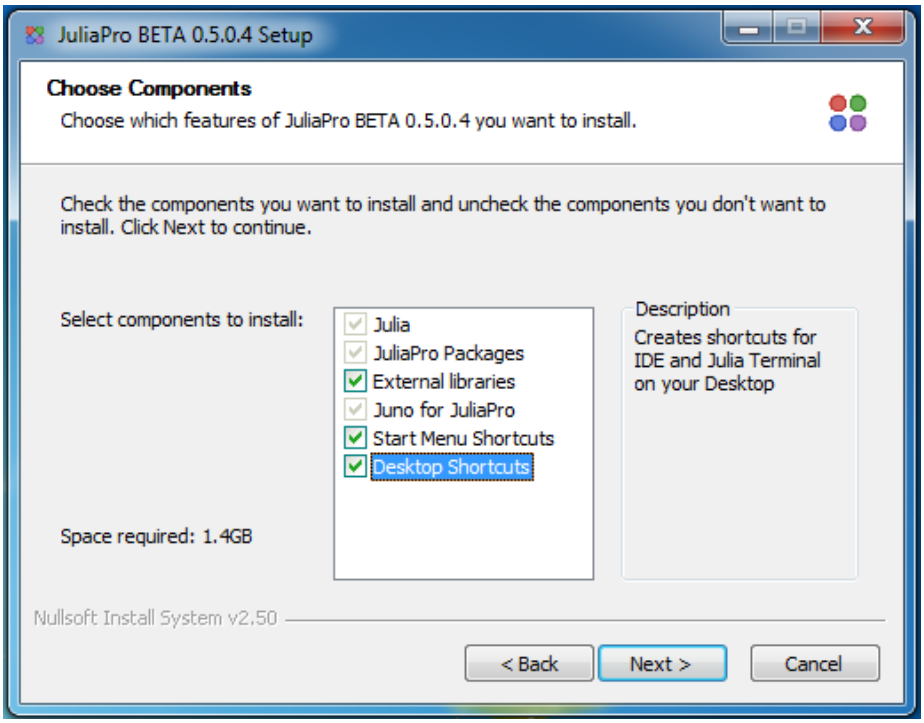
The next listed component includes "Juno for JuliaPro". Selection of this component will install a copy of the Atom editor onto which various JuliaPro specific enhancements are continually being added.



The subsequent component is the entry for "Start Menu Icons". By unselecting this entry, no entries for JuliaPro will be added to the Windows Start Menu.



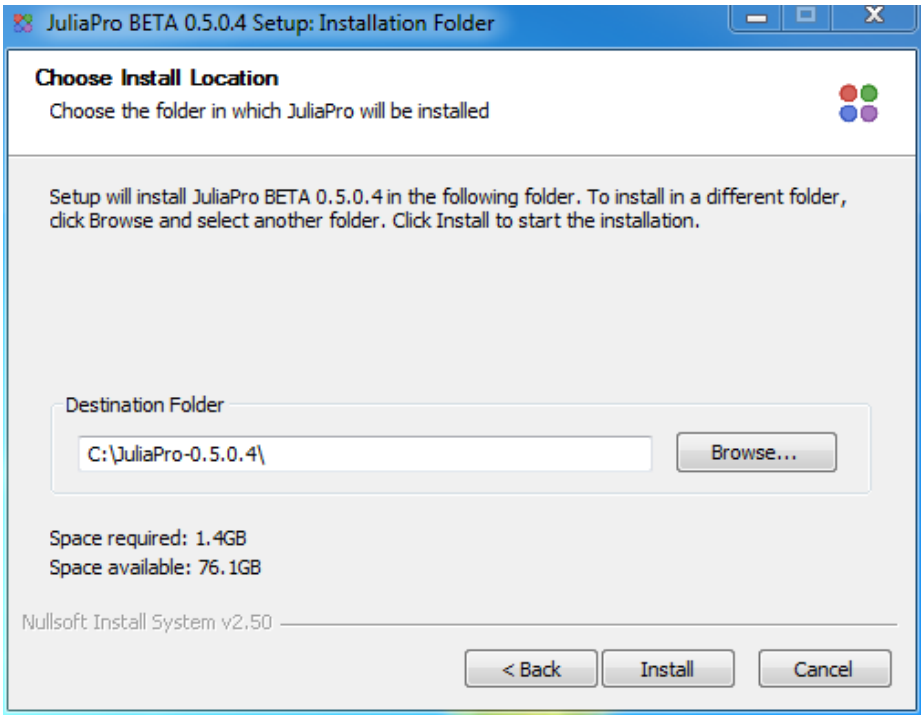
The final component is the entry for "Desktop Shortcuts". By unselecting this entry, no desktop shortcut icons linking to various JuliaPro components are created.



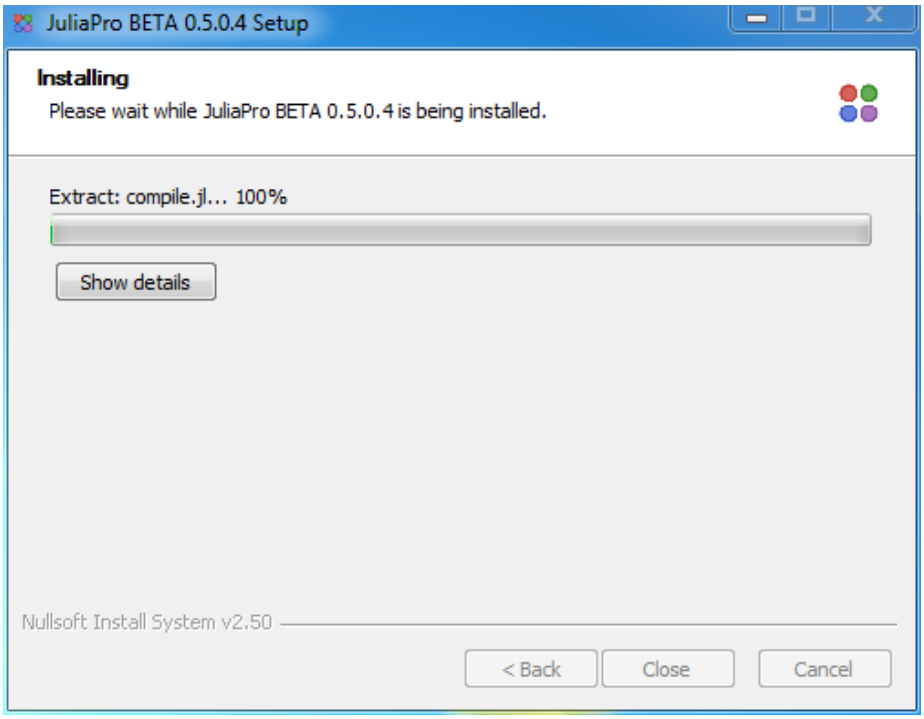
Upon selecting your desired components, click "Next" to proceed.

Next, select the folder into which you wish to install JuliaPro. The selection must be a directory into which the current user has read/write privileges. The installer provides an appropriate default location when either the "All Users", "Current User" options were selected.

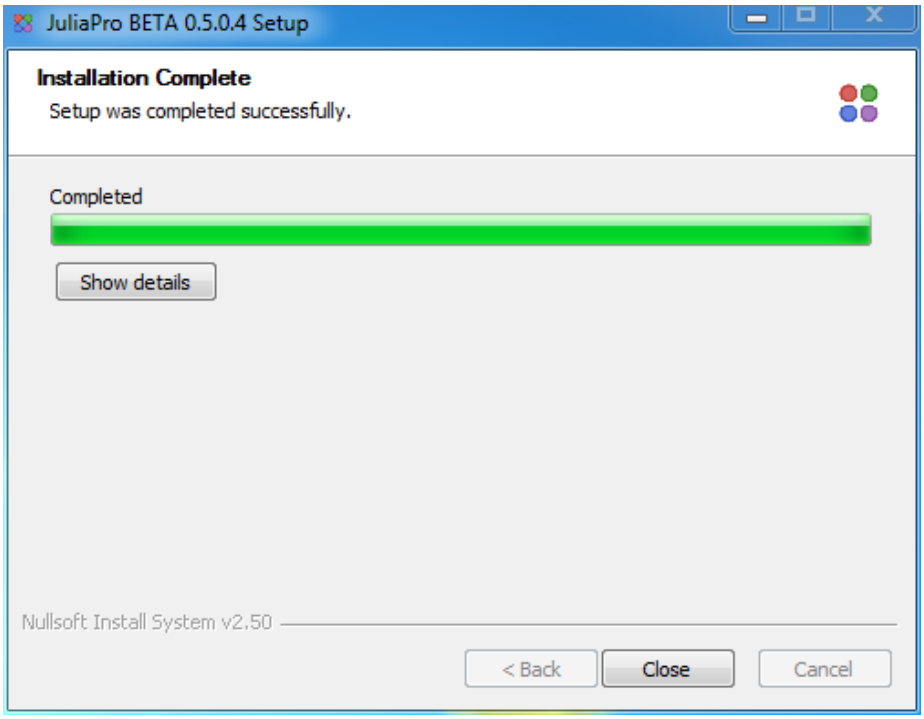
If the "Shared Drive" option was selected by a non-Administrative user, or an Administrative user selects the "All Users" option, then you will likely wish to click the "Browse..." button in order to select a shared network drive location into which to install JuliaPro. Installing JuliaPro to a shared network drive location is useful for teams that plan to share access to a single Julia Professional installation among multiple user accounts on separate machines in the same Active Directory domain.



The following screenshot shows the JuliaPro installer during its installation phase.



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



NOTE: Setting access permissions on your JuliaPro installation for shared installations.

For shared network drive installations, if the directory into which JuliaPro was installed is currently accessible by multiple users, then an Administrator may desire to set access permissions for their JuliaPro installation to be read-only for non-Administrative users.

By setting the JuliaPro installation to be read-only, only administrative users will be able to perform operations via Julia's built-in Pkg package manager. Operations such as `Pkg.add`, `Pkg.update`, and `Pkg.rm` can only be performed by users with write access to the `pkgs-0.5.0.4/v0.5` and `pkgs-0.5.0.4/lib` directories of the Julia Professional installation.

Restricting write access to these directories to a single administrative user will ensure a consistent state of packages for all users of a shared installation.

To prevent unnecessary precompilation of Julia packages from being triggered by separate users, all users accessing a shared drive installation of JuliaPro from a mapped network drive must have the shared drive mapped to the same network drive letter and absolute path.

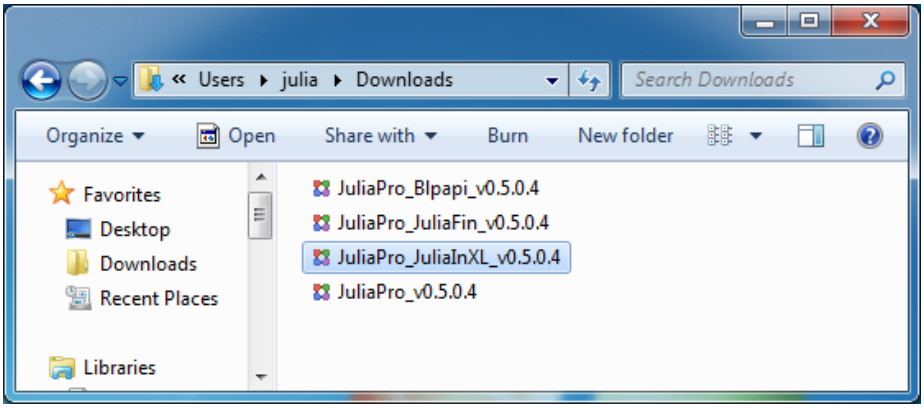
NOTE: Copying of Atom Configuration Information on First Launch of Juno.

For shared network drive installation, the first time a user launches Juno from

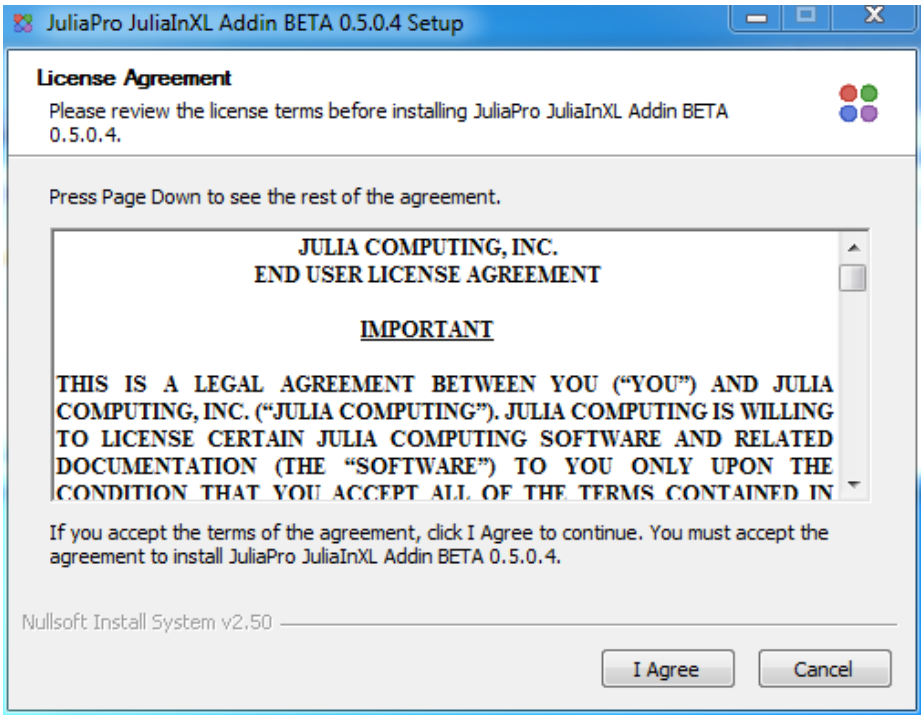
their client machine, a `.atom` configuration folder will be copied from the Julia Professional installation into the directory `%HOMEDRIVE%\%HOMEPATH%\JuliaPro_Juno_0.5.0.4`. This directory stores user specific configration information for use of Juno for JuliaPro.

6.2 3.2. Installing JuliaInXL for JuliaPro

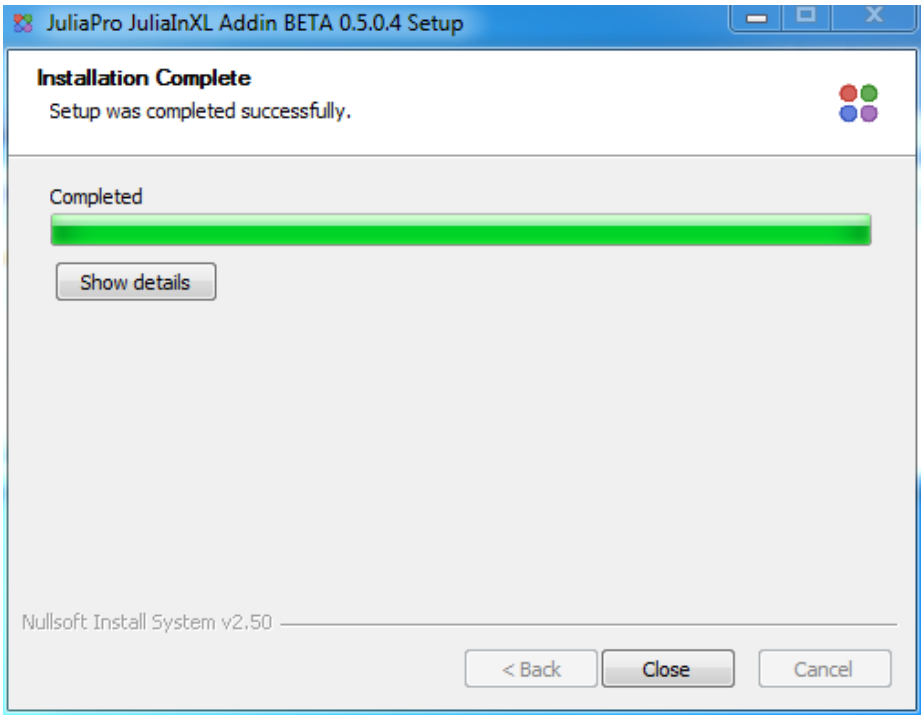
JuliaInXL for JuliaPro can be installed using the provided executable installer as shown below.



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

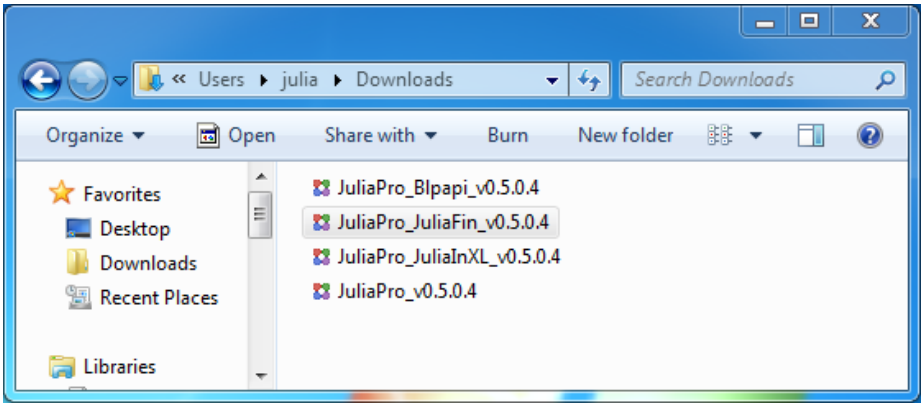


Upon completion of the installer, press close to exit the installer.

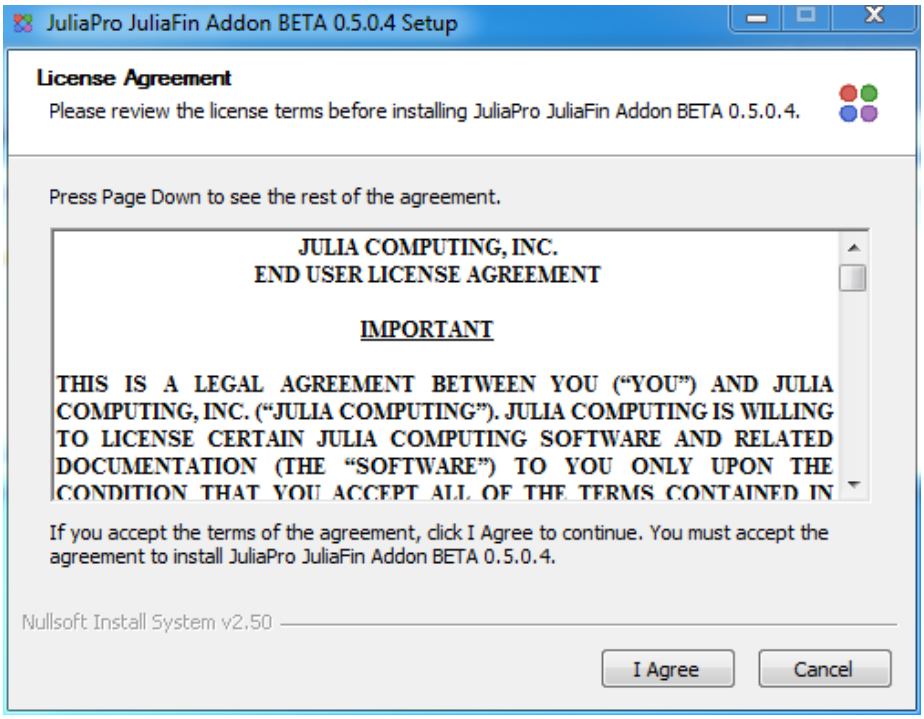


6.3 3.3. Installing JuliaFin for JuliaPro

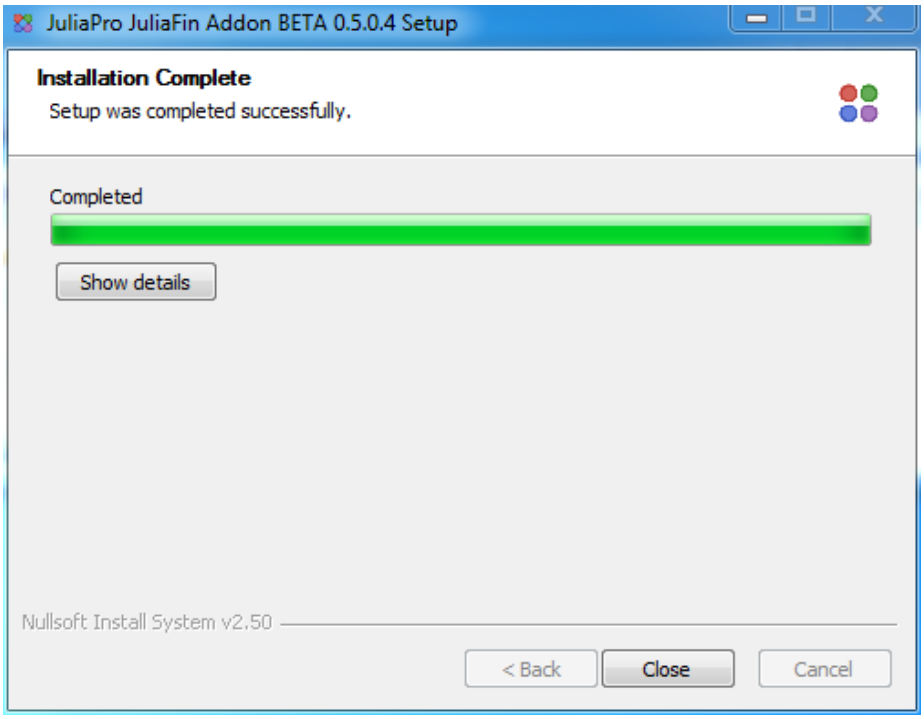
JuliaFin for JuliaPro can be installed using the provided executable installer as shown below.



Upon launching JuliaFin installer, you will be presented with the JuliaPro Software License Agreement. After reading through the terms mentioned in the agreement, click “I Agree” if you accept the terms of the license and proceed with the installation.



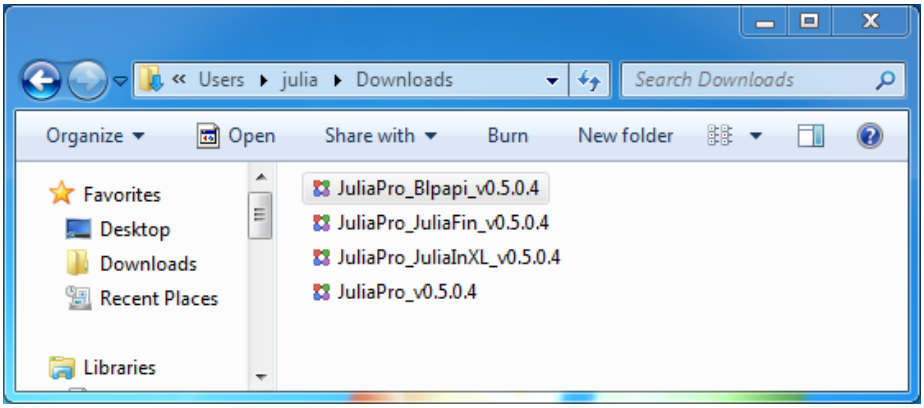
Upon completion of the installer, press close to exit the installer.



For instructions on usage of JuliaFin and Miletus, please see the documentation installed as part of the JuliaFin package.

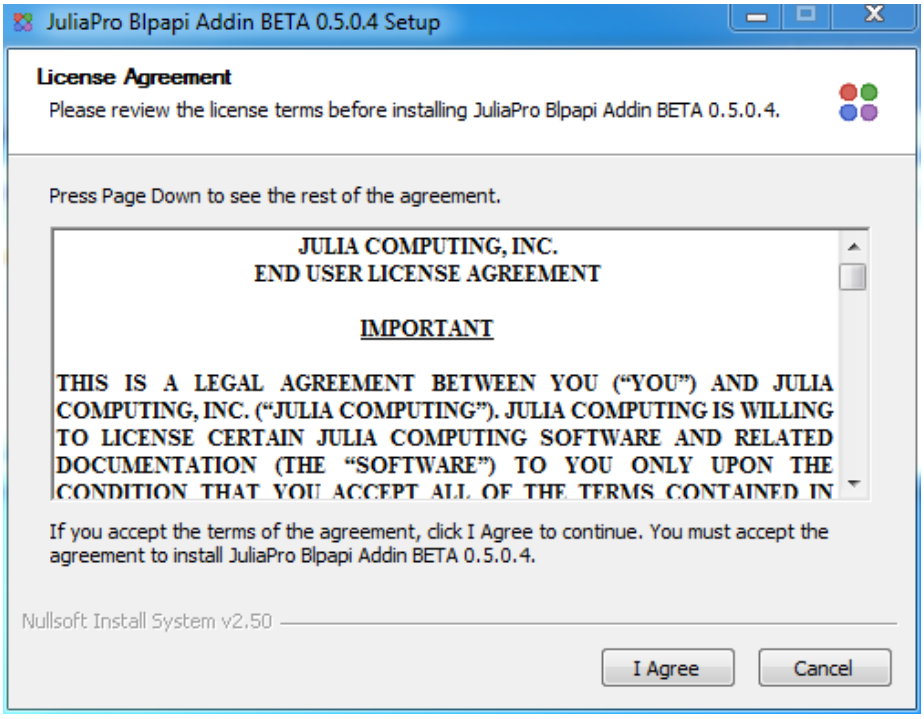
6.4 3.4. Installing Blpapi for JuliaPro

Blpapi for JuliaPro can be installed using the provided executable installer as shown below.

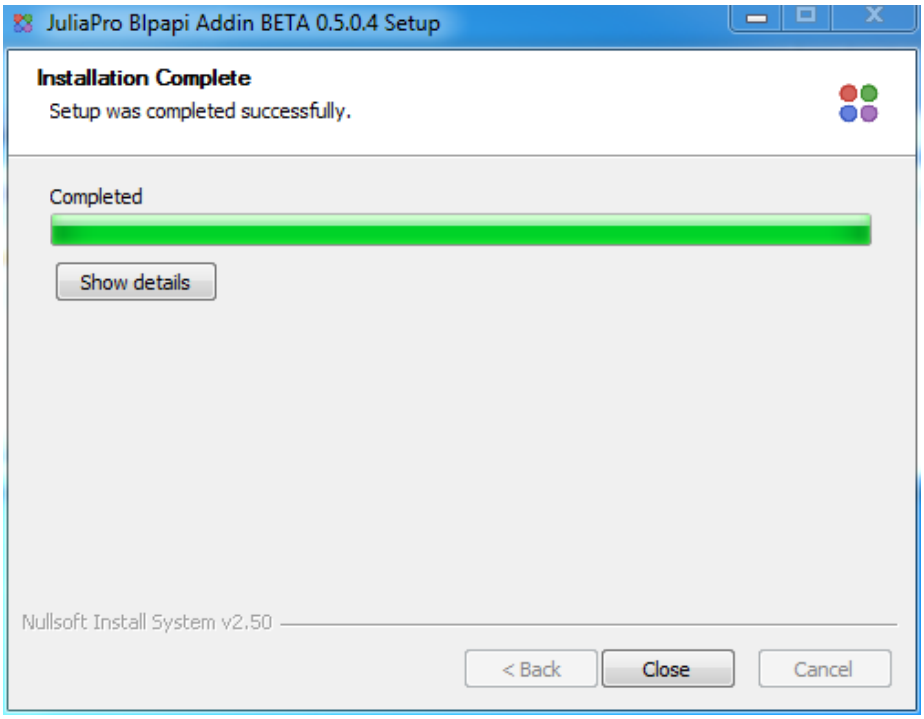


Upon launching Blpapi installer, you will be presented with the JuliaPro Software License Agreement. After reading through the terms mentioned in the agreement,

click “I Agree” if you accept the terms of the license and proceed with the installation.



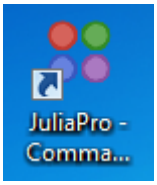
Upon completion of the installer, press close to exit the installer.



For instructions on usage of Blpapi.jl, please see the documentation installed as part of the Blpapi package.

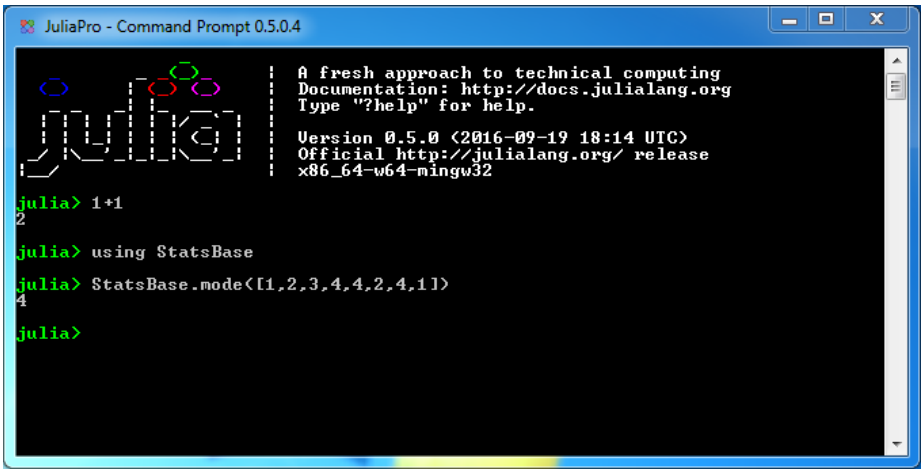
7 4. Using the JuliaPro Command Prompt

If the Desktop Shortcuts components were selected during the installation process, then you should have a Julia Professional Command Prompt icon on your desktop. Double-Click on the icon to start a JuliaPro terminal.



If the Start Menu Shortcuts components were selected during the installation process, then you should have an entry for the JuliaPro Command Prompt located under "Start -> JuliaPro -> JuliaPro Command Prompt"

Upon launching a new command prompt window, a Julia REPL 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.

7.1 4.1. Configuring PyCall and IJulia for use with your own Python installation

If your system already has an installation of Python, Jupyter, Matplotlib and the ipywidgets package for Python, then it is possible to configure your JuliaPro installation to use those packages instead of the Python distribution that is bundled with the "External Dependencies" option in the JuliaPro installer. While it is possible to use JuliaPro with a pre-existing Python installation, Julia Computing does not currently provide support for use of JuliaPro with pre-existing Python installations as part of a JuliaPro support subscription.

To configure your JuliaPro installation to use your pre-existing Python installation, perform the following steps:

1. Open the `juliarc.jl` file that is present in `<path-to-JuliaPro>\Julia-0.5.0\etc\julia\juliarc.jl`
2. Add the following lines to the file, replacing the entries for `<Path_to_Jupyter.exe>` and `<Path_to_Python.exe>` with the paths to the `juypter.exe` and `python.exe` executables for your particular Python installation.

```
julia if !(haskey(ENV, "JUPYTER")) ENV["JUPYTER"] = <Path_to_Jupyter.exe>
end if !(haskey(ENV, "PYTHON")) ENV["PYTHON"] = <Path_to_Python.exe>
end
```

3. Launch a JuliaPro command prompt as described in the previous section.

4. Execute the following commands in your Julia command prompt window:

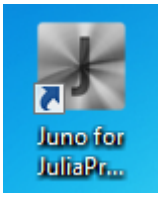
```
julia julia> Pkg.build("PyCall"); julia> Pkg.build("IJulia");
```

Your JuliaPro installation should now be configured to use your pre-existing Python and Jupyter installations.

8 5. Using Juno for JuliaPro

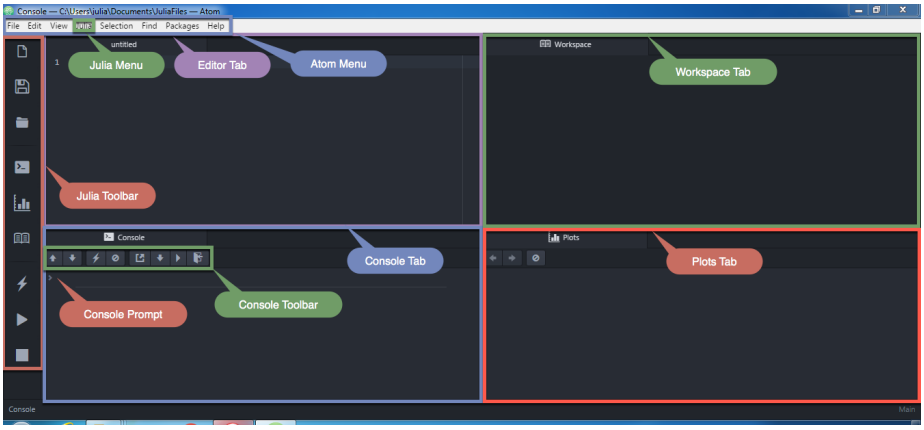
8.1 5.1 Launching Juno for JuliaPro

If the Desktop Shortcuts components were selected during installation of the JuliaPro IDE, then you should have a Juno for JuliaPro icon on your desktop. Double-Click on the icon to start a Juno for JuliaPro session.



If the Start Menu Shortcuts components were selected during the installation process, then you should have also an entry for the JuliaPro Command Prompt located under "Start -> JuliaPro -> Juno for JuliaPro 0.5.0.4"

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



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

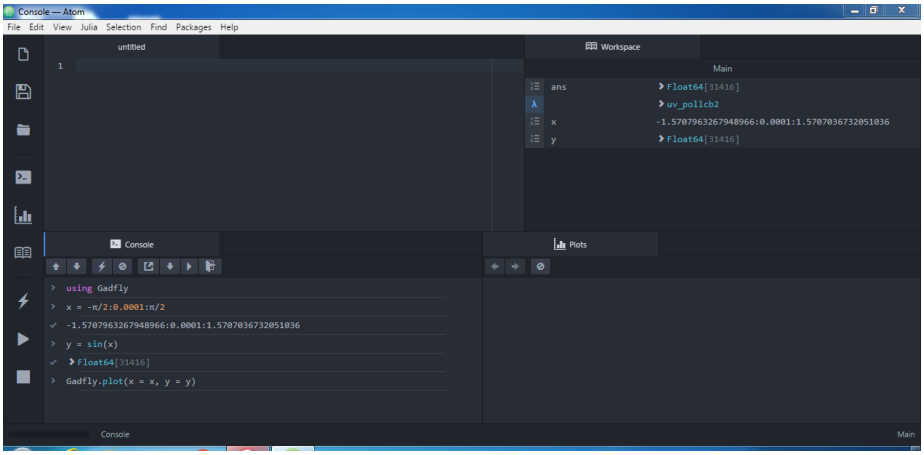
When getting started with the use of Julia in the Atom IDE, there are four primary panes where users will spend the majority of their time and effort:

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

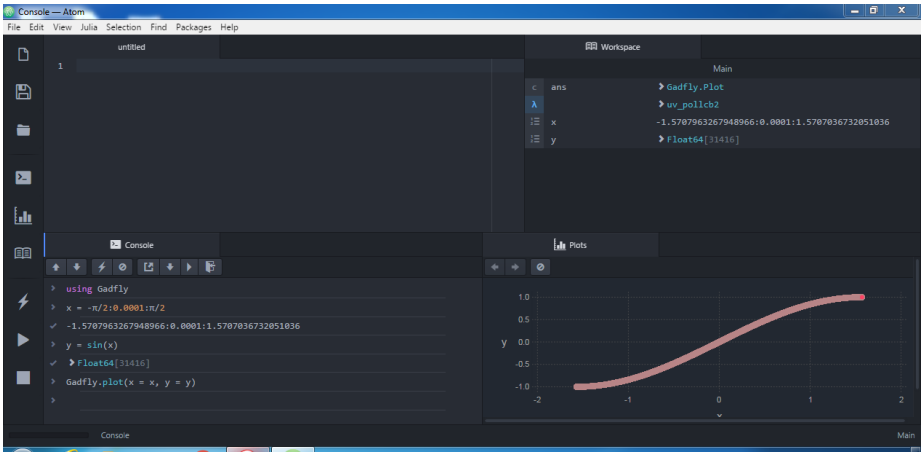
Within the Editor tab, users can create, edit, open and save Julia source code files. Juno also allows for code to be executed, and associated results displayed, directly inline within the Editor tab. The Console tab provides an interactive Julia command prompt for immediate command execution, and the 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.

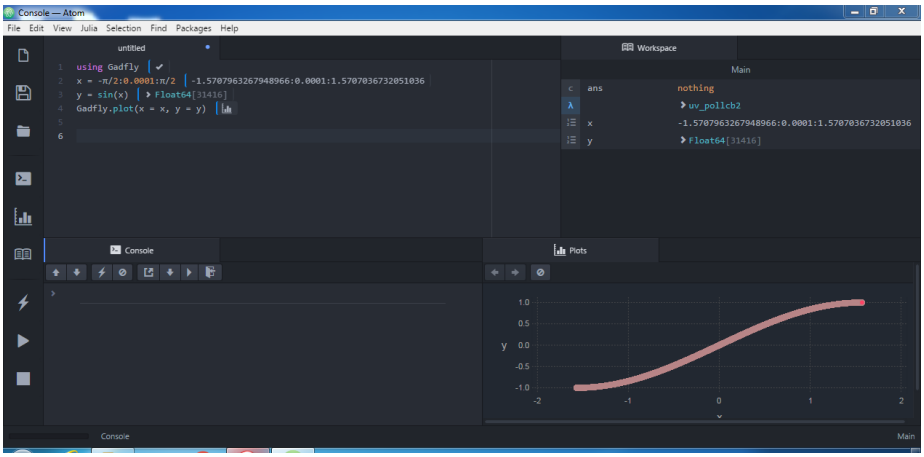


A resulting plot is displayed within the Plots tab.

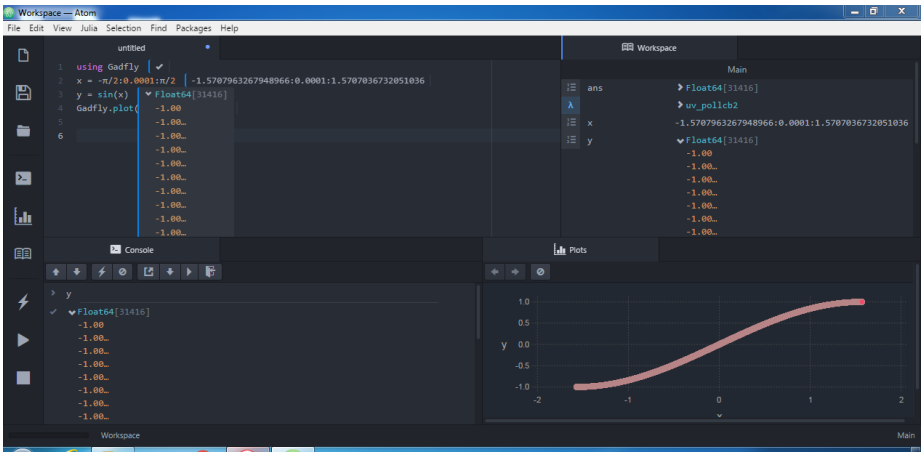


While some users' preferred workflow involves developing and executing short interactive snippets of code in the Console tab, and typing longer segments of code within the Editor tab, Juno also provides functionality for executing code directly within the Editor tab.

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.

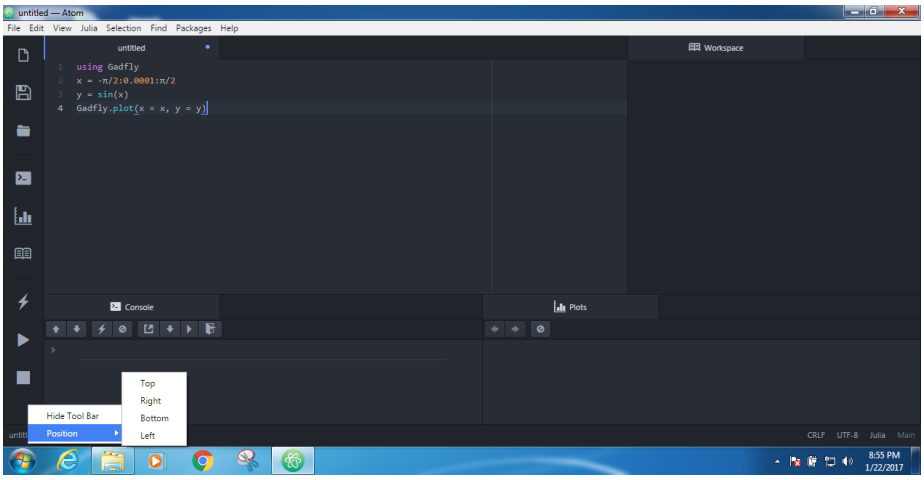
8.3 5.2.1 Using the Julia Toolbar

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.

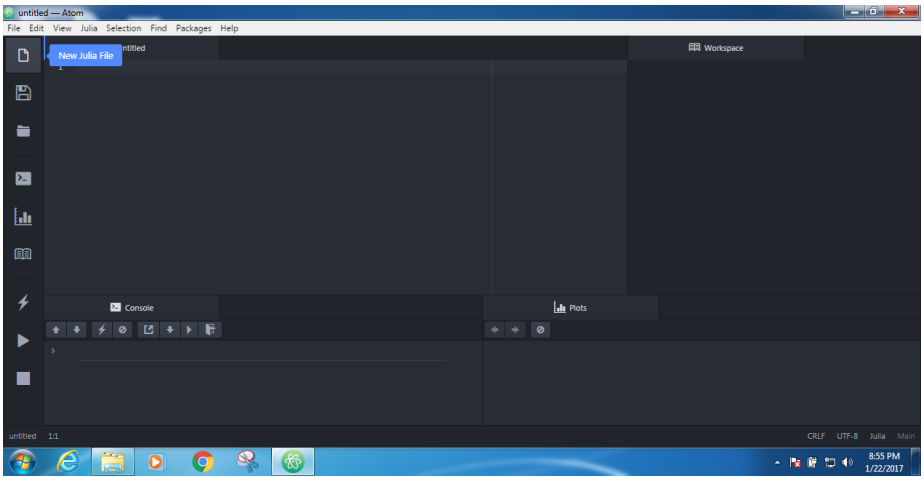


8.5 5.2.1.2 Toolbar Contents

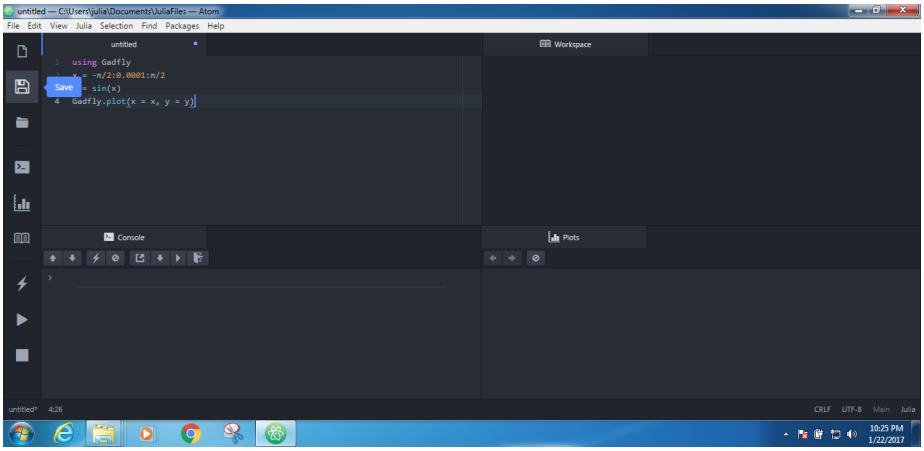
The Julia Toolbar contains buttons that represent the following functions (listed in vertical order from top to bottom when toolbar is in its default left location):

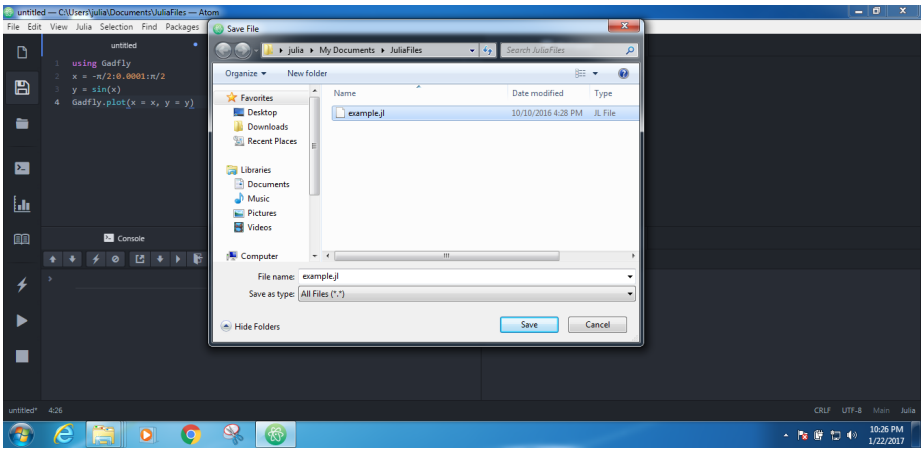
- New Julia File
- Save
- Open File
- Open Console
- Show Plots
- Run Block
- Run File
- Stop Julia

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

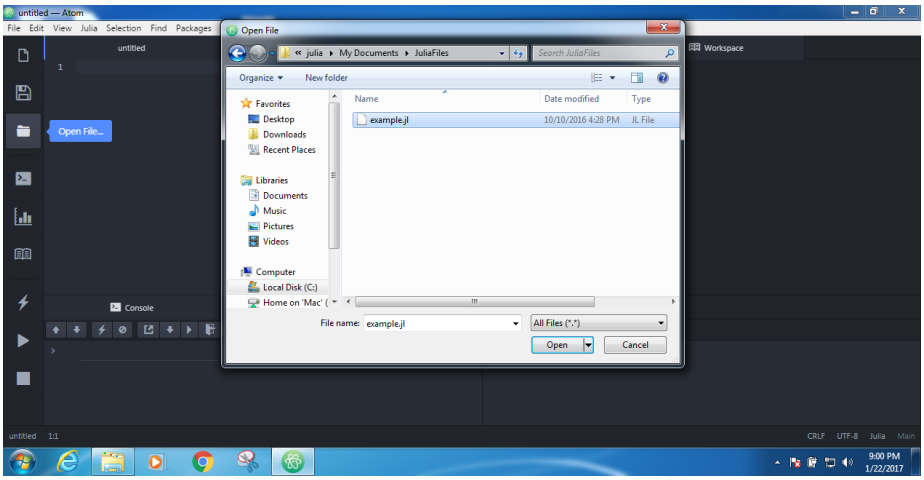


The “Save” button will open a pop-up window allowing the user to save the currently active Julia source file to a user defined location.

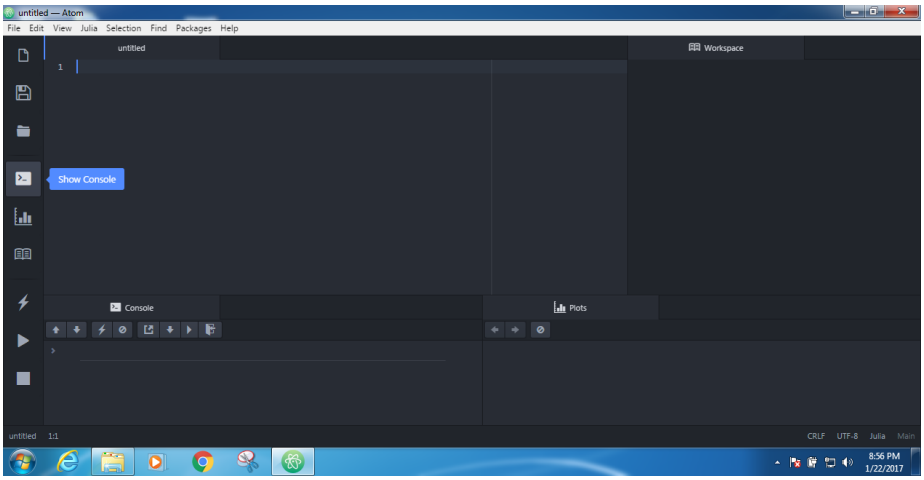


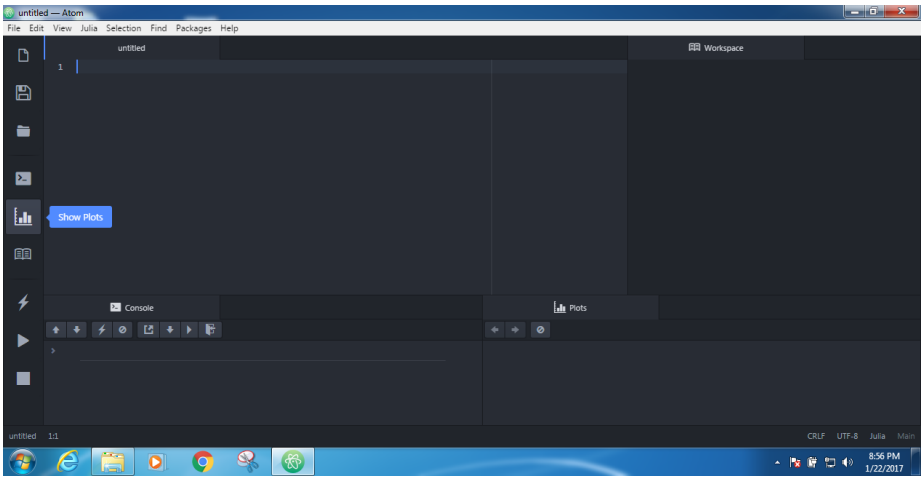


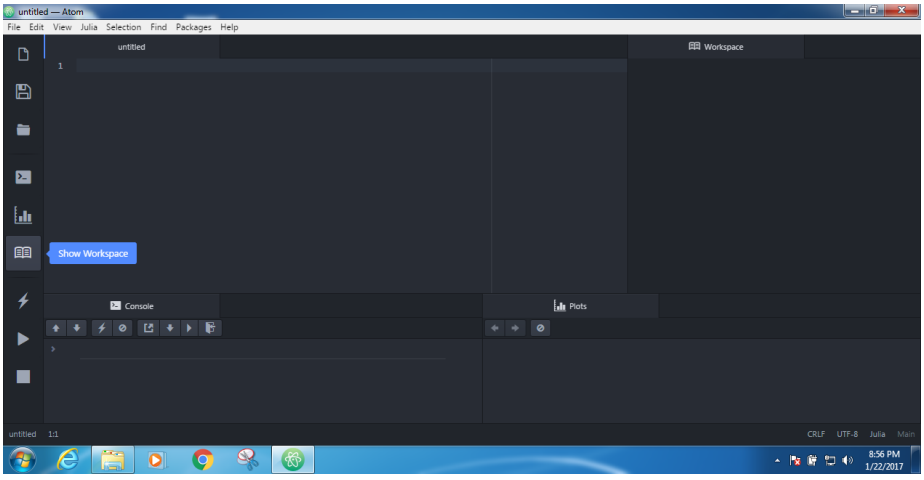
Similarly, the "Open File…" button allows the user to select a source file that they would like to open in the current editor pane.



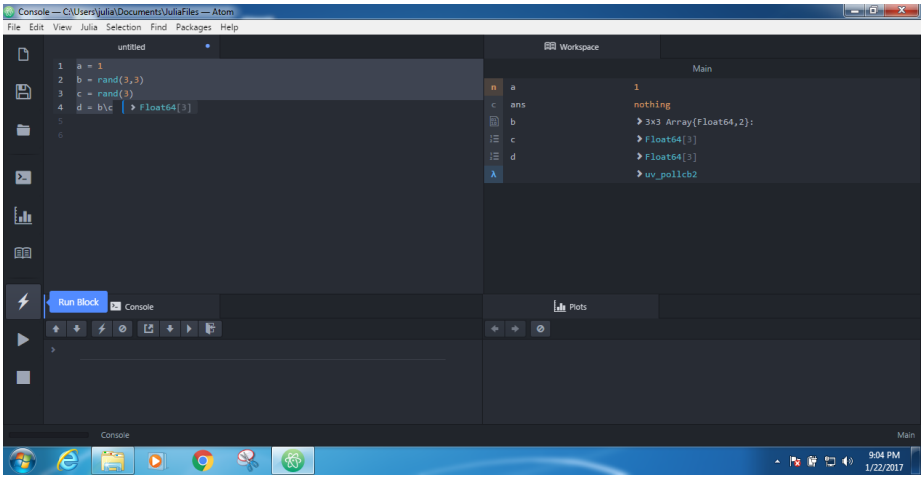
If any of the Console tab, the Plots tab, or the Workspace tab has been closed, then the "Show Console", "Show Plots", or "Show Workspace" buttons can be used to re-open those particular tabs.


untitled — Atom
File Edit View Julia Selection Find Packages Help
untitled
Workspace
Show Console
Console
Plots
untitled 1:1
CRLF UTF-8 Julia Main
8:56 PM
1/22/2017

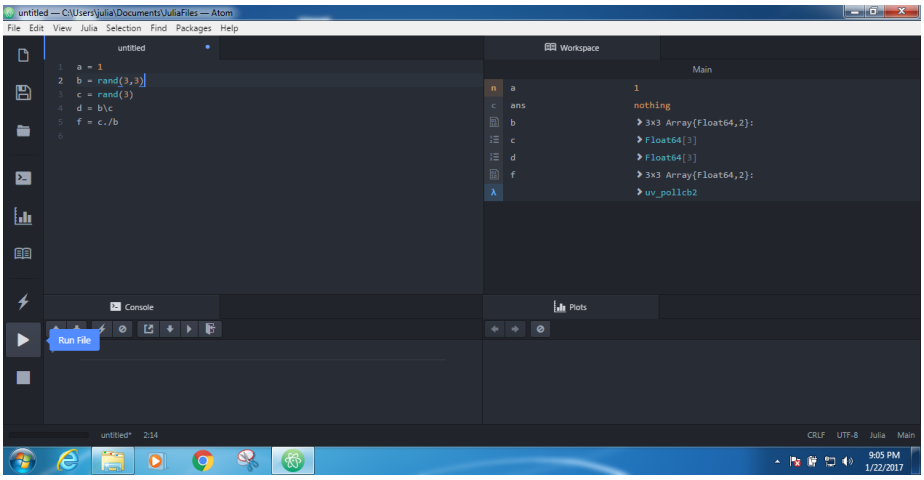

untitled — Atom
File Edit View Julia Selection Find Packages Help
untitled
Workspace
Show Plots
Console
Plots
untitled 1:1
CRLF UTF-8 Julia Main
8:56 PM
1/22/2017


untitled — Atom
File Edit View Julia Selection Find Packages Help
untitled
Workspace
Show Workspace
Console
Plots
untitled 1:1
CRLF UTF-8 Julia Main
8:56 PM
1/22/2017

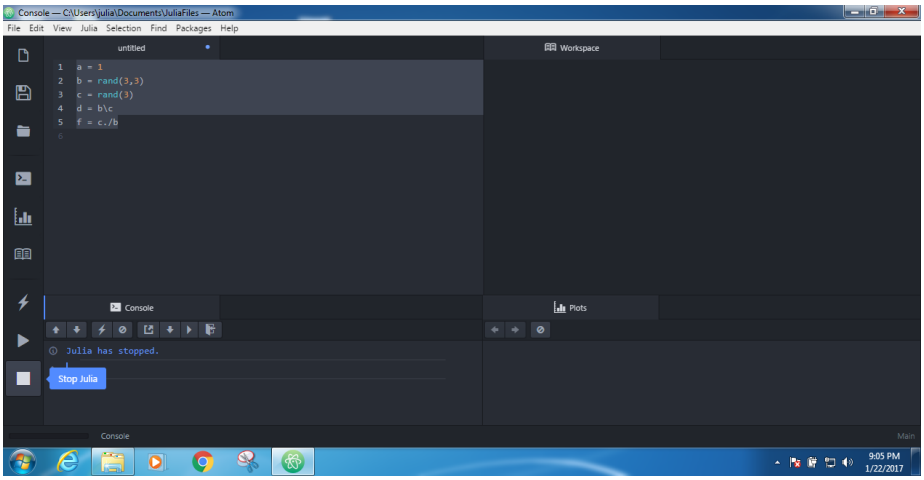
The “Run Block” button will execute a series of highlighted statements from within the editor. This command can also be accessed via the keyboard shortcut Ctrl-Enter.



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 Ctrl-Shift-Enter.



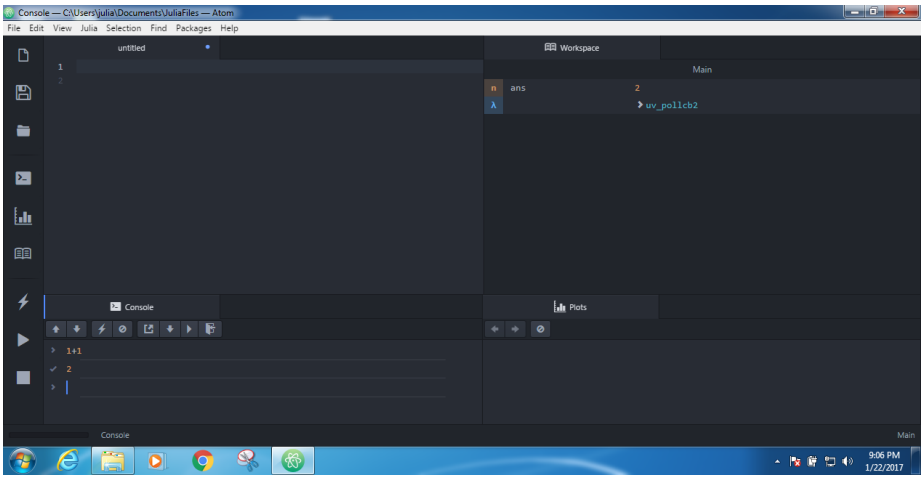
The “Stop Julia” button will stop the currently active Julia kernel.



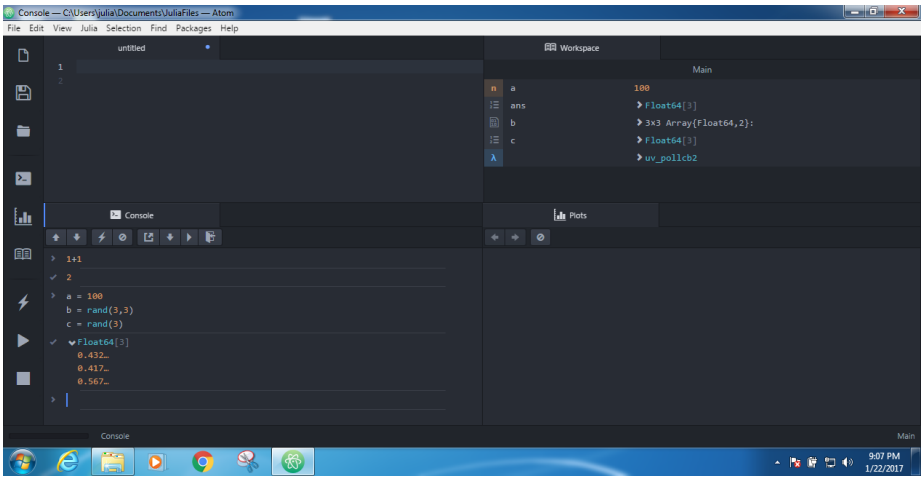
8.6 5.2.2 Using the Console Tab

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.

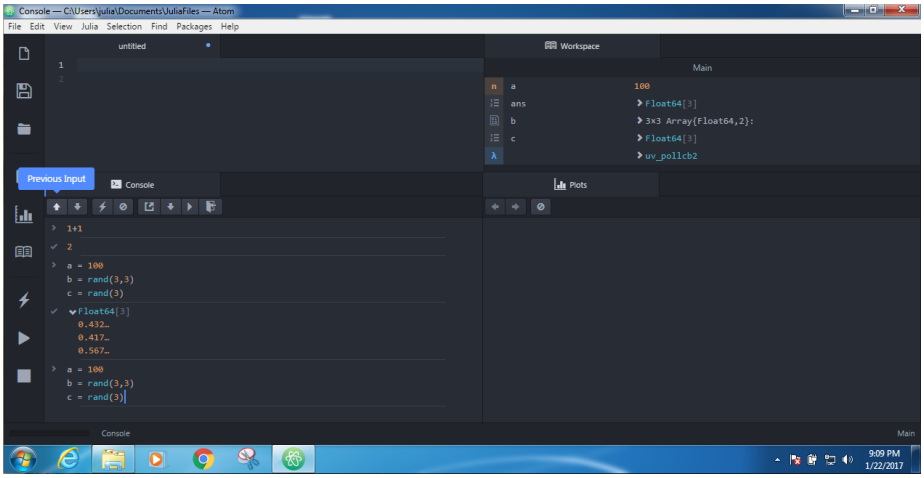
Executing a single Julia statement within the Console tab requires either pressing Enter upon completing the entry of the command, or pushing the Run button in the Console toolbar.

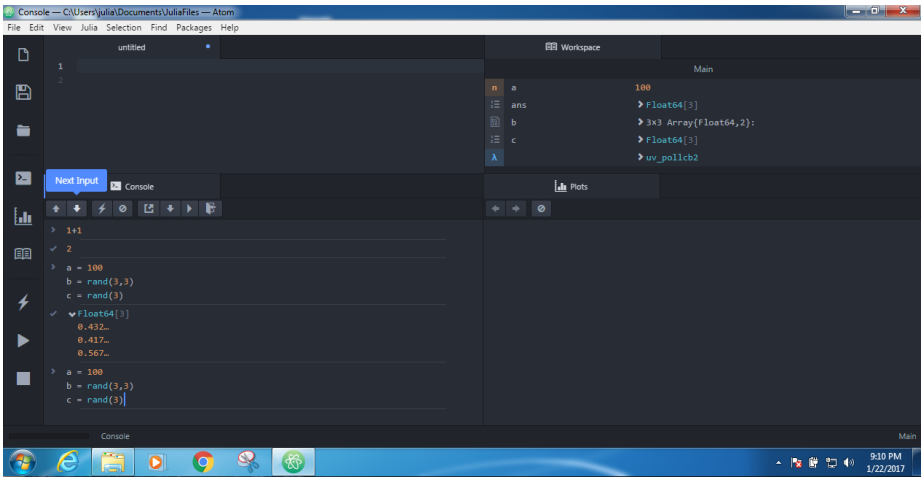


To enter a multiline command within the Console tab, press Shift+Enter upon the completion of a line in the Console. Subsequently pressing the Enter key or pushing the Run button will execute all of the commands in a multiline statement.



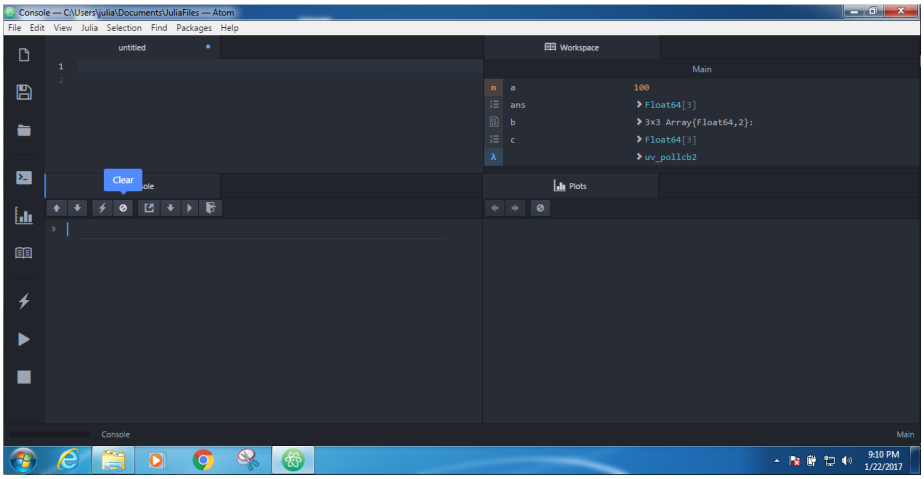
Scrolling through previously entered commands in the Console tab can be accomplished via either the up and down keys on your keyboard, or via the Previous Input and Next Input buttons in the Console toolbar.





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

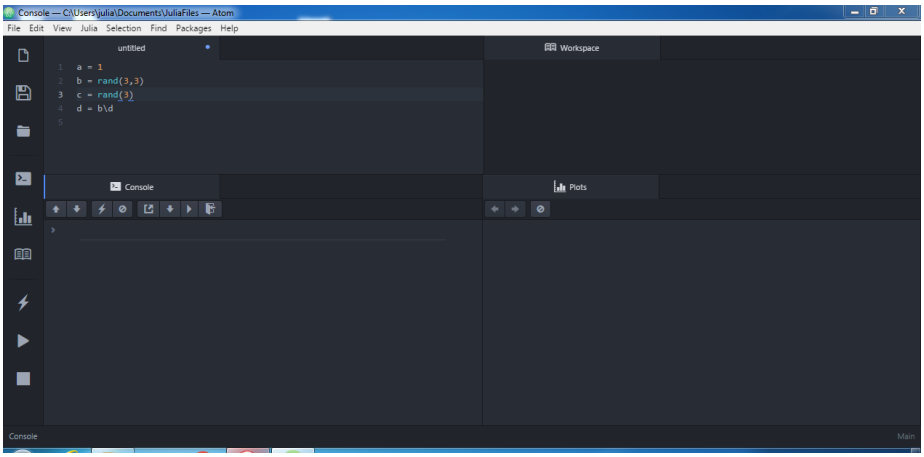
The currently displayed text within the Console tab can also be cleared via the Clear button.



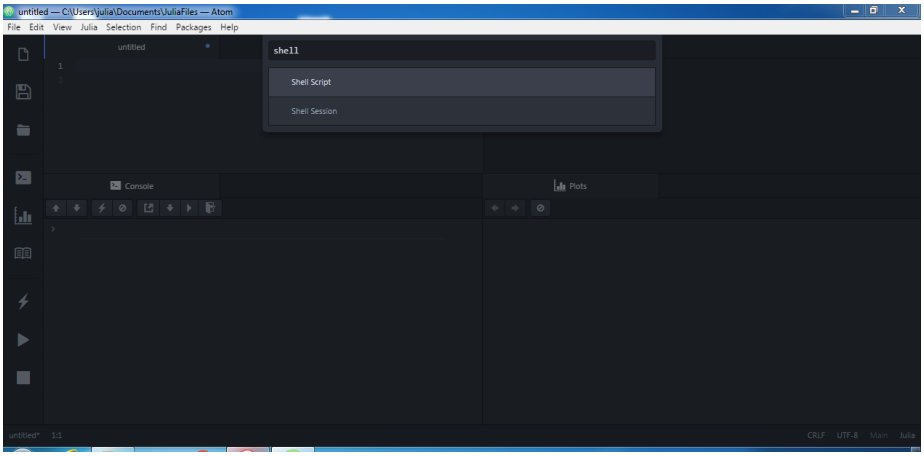
The remaining buttons in the Console toolbar are used for the debugging of Julia code and are discussed in the section Debugging Using the Console and Editor Tabs below.

8.7 5.2.3 Using the Editor Tab

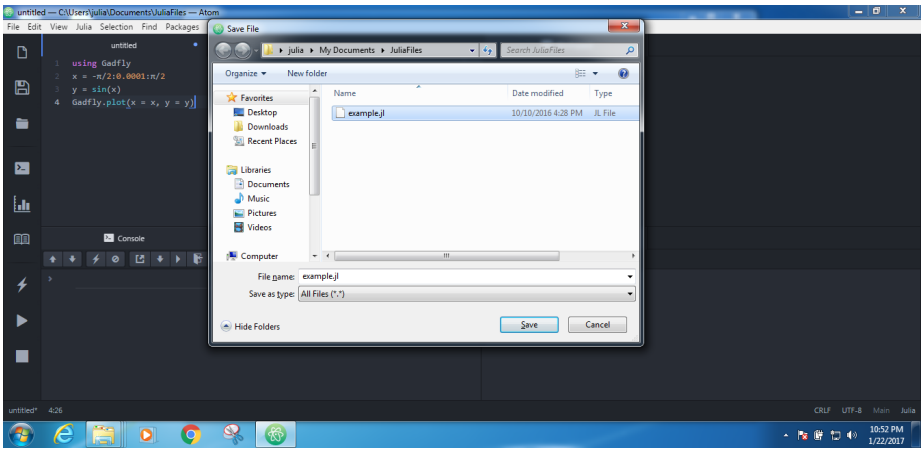
The Editor tab can be used for developing any Julia scripts, modules, or packages that can be saved as *.jl files.



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.

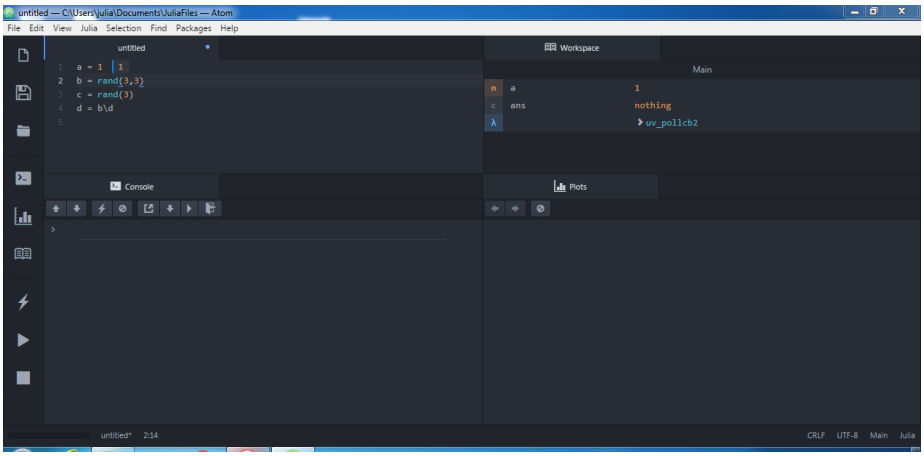


Upon editing the current source file, any edits can be saved by either pressing the Ctrl+S keystroke combination, pressing the Save button in the main Julia toolbar, or selecting "File -> Save" from the Atom menu bar.

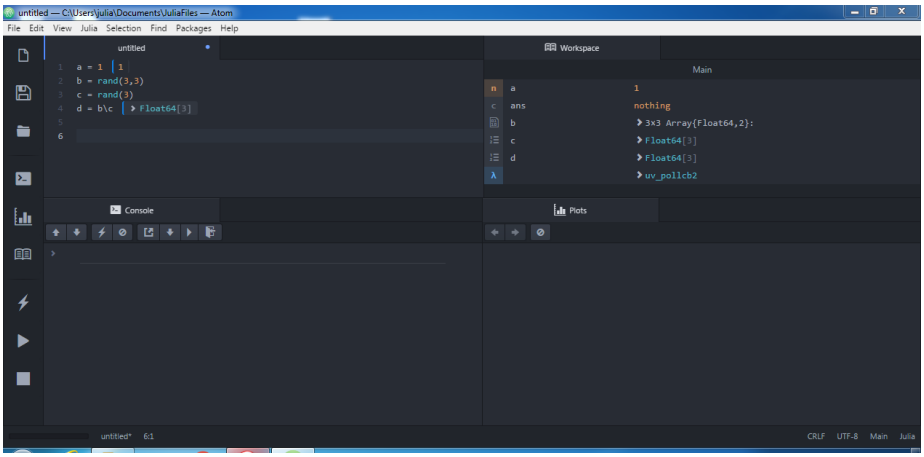


As stated previously, in addition to being used just for editing code, Julia source code can also be executed directly from within the Editor tab.

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.

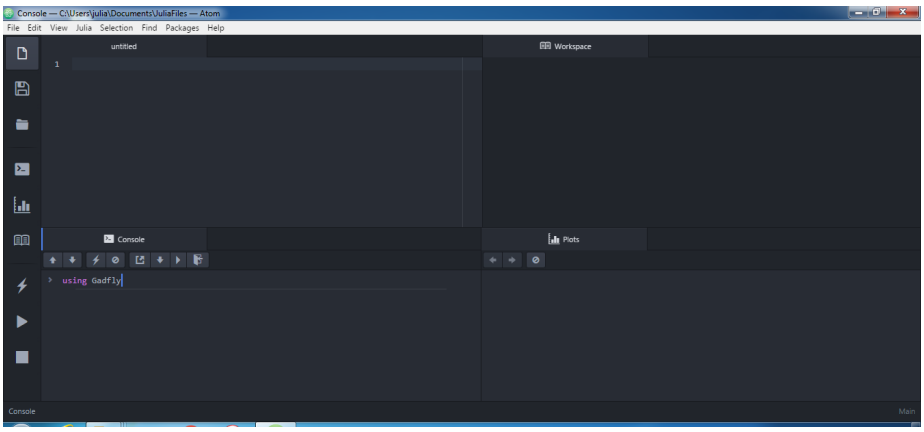


8.8 5.2.4 Using the Plots Tab

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

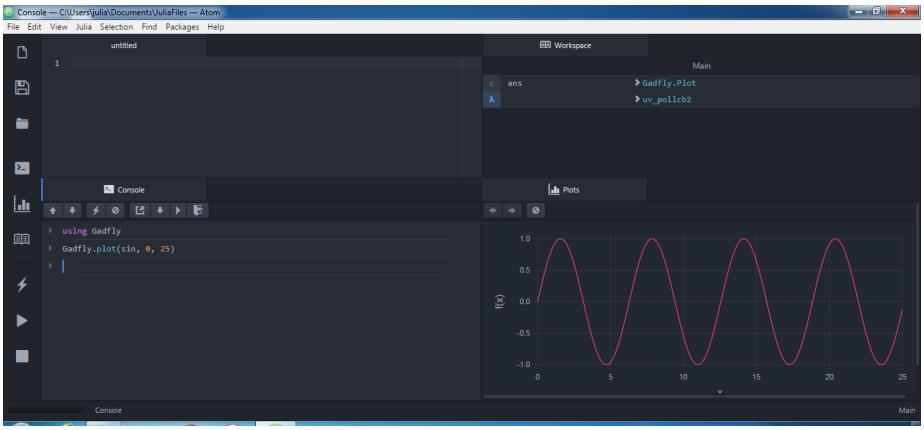
To create a plot using Gadfly, one first needs to load the Gadfly package as follows:

```
> using Gadfly
```



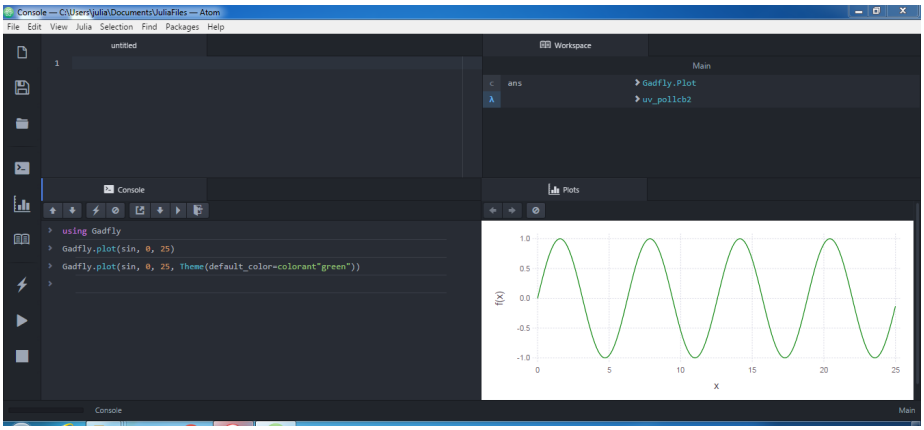
One can create a simple plot of a sine curve as follows:

```
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.

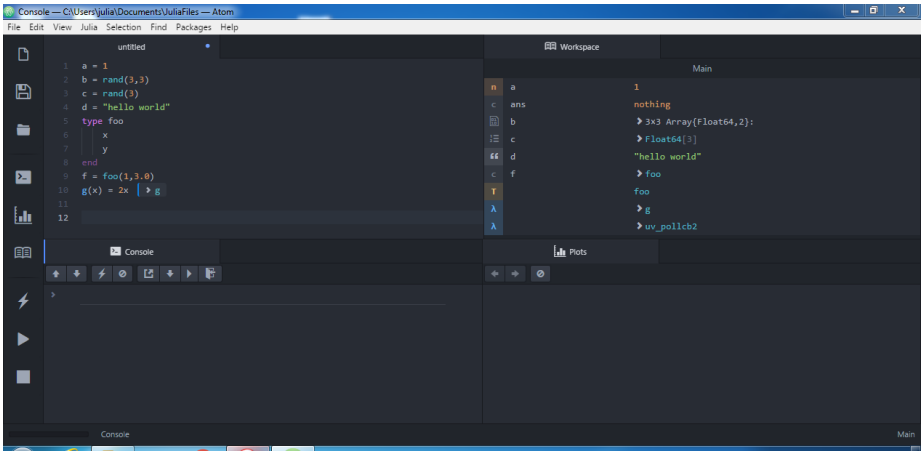
See the Gadfly documentation for its full list of options for customizing a plot.

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.

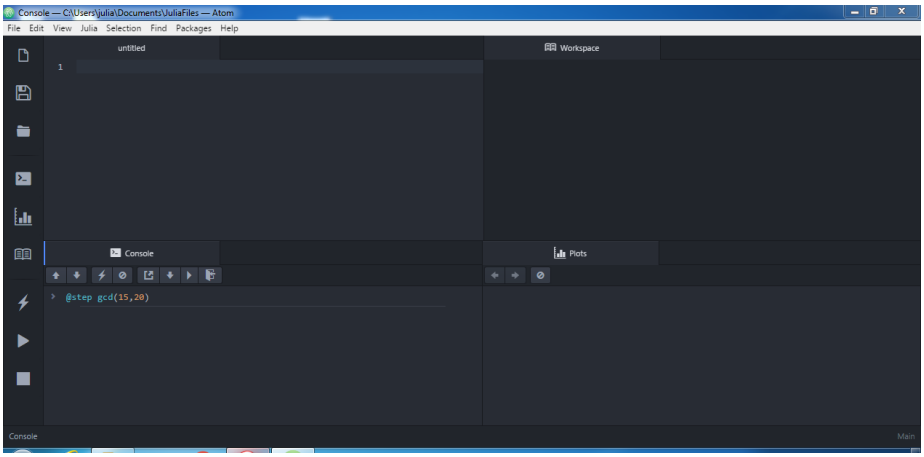


8.10 5.2.6 Debugging Using the Console and Editor Tabs

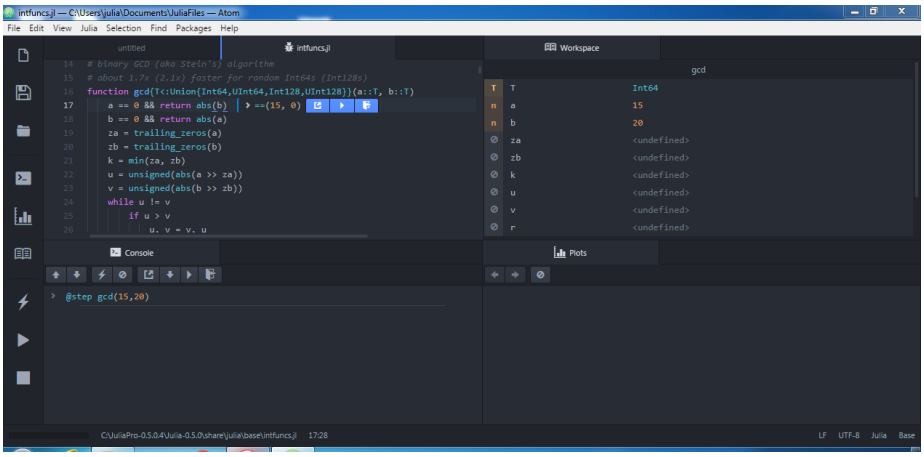
The final four buttons in the Console Toolbar allow for use of functionality from Julia's Gallium debugger from within Juno. These four buttons include:

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

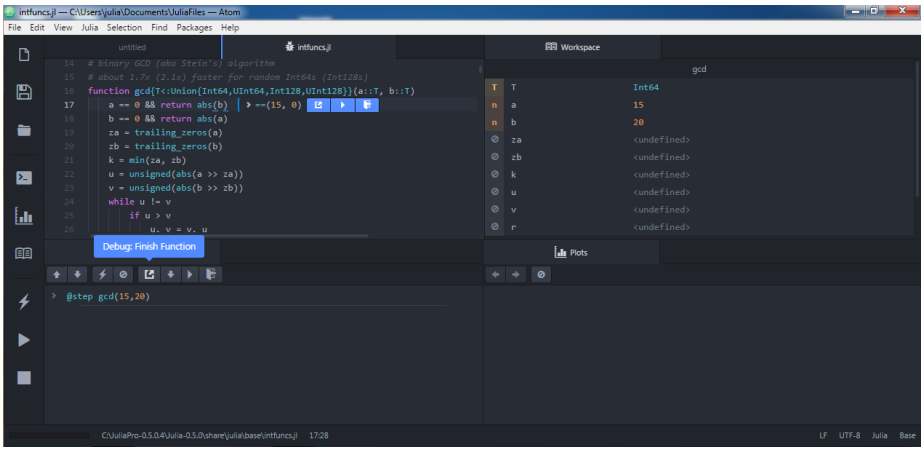
Use of debugging functionality from within Juno is associated with use of the `@step` macro.



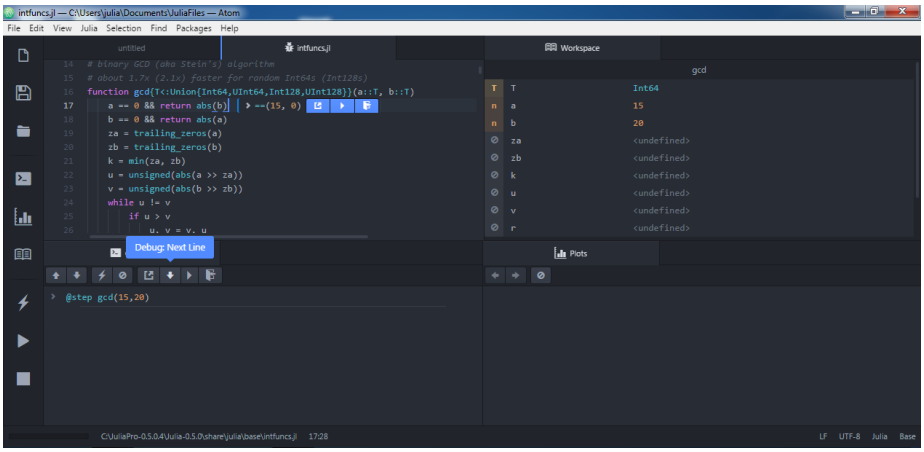
Upon executing a statement preceded by the `@step` macro, the source file containing the relevant method will open within an editor pane, and a small pop-up toolbar is presented at the line where a Gallium breakpoint has been hit. The Workspace window will also display variables in the scope of the method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



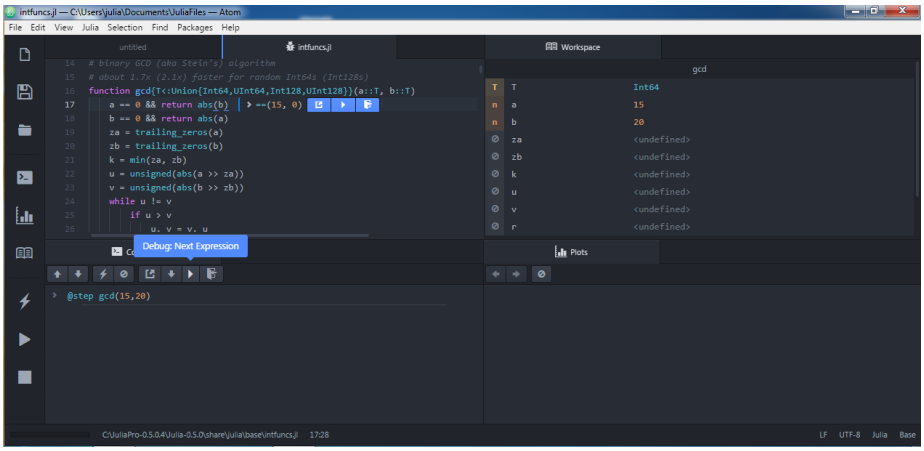
The "Debug: Step to Next Line" button executes the active line in the current source file and stops at the subsequent line.



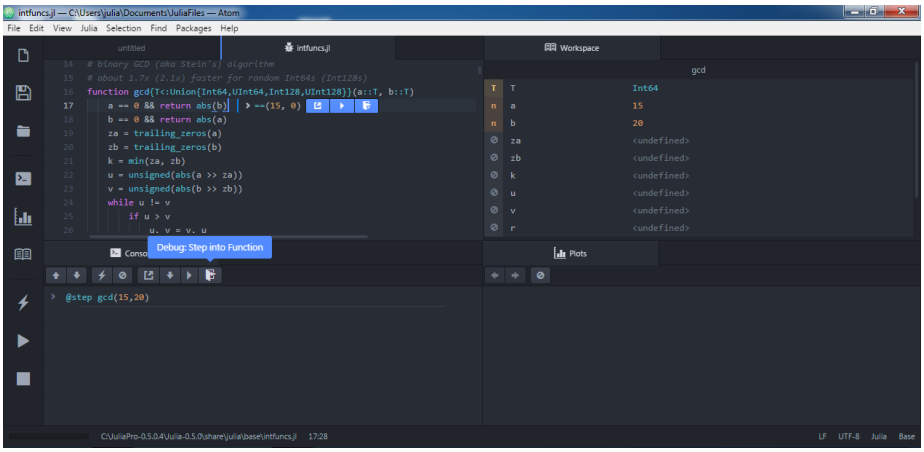
The "Debug: Finish Function" button executes all statements included in the function where the currently active debugger stop is located and returns to the scope from which the current function was called.



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.

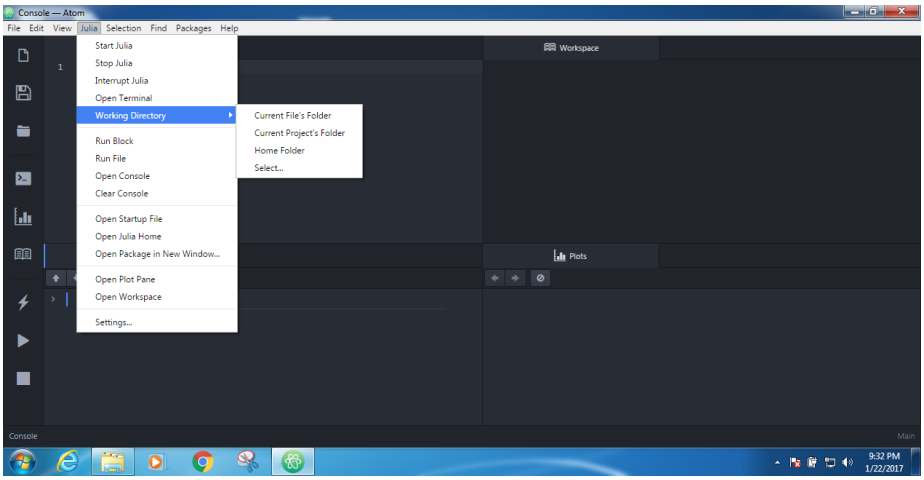


The "Debug: Step to Next Expression" button advances execution of the program to the next expression in the active line of the current source file. Use of this button allows one to proceed from one subexpression to another on a single line, and progressively execute and inspect nested expressions on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

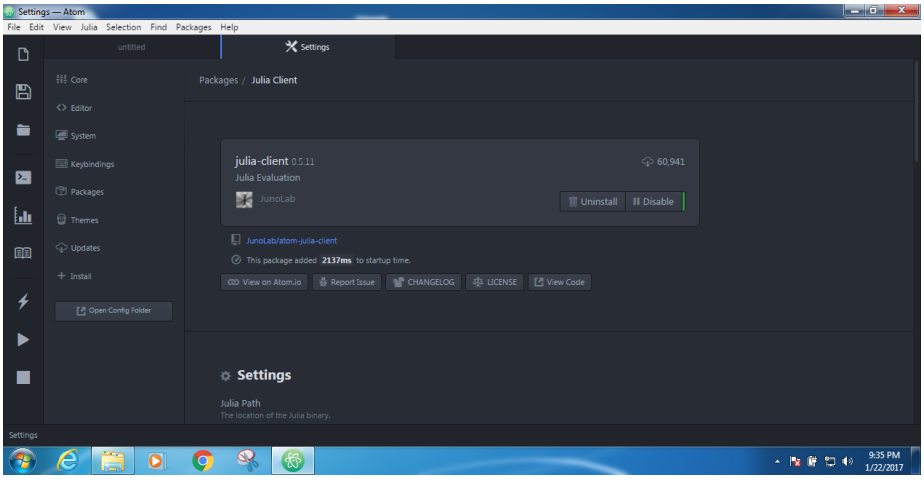


The Julia menu includes the following entries:

- 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

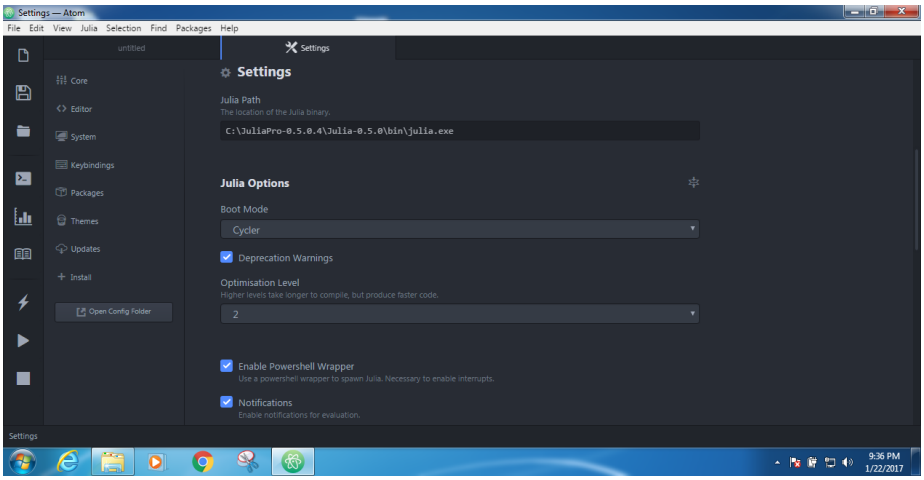
8.12 5.3.1 Julia Settings Tab

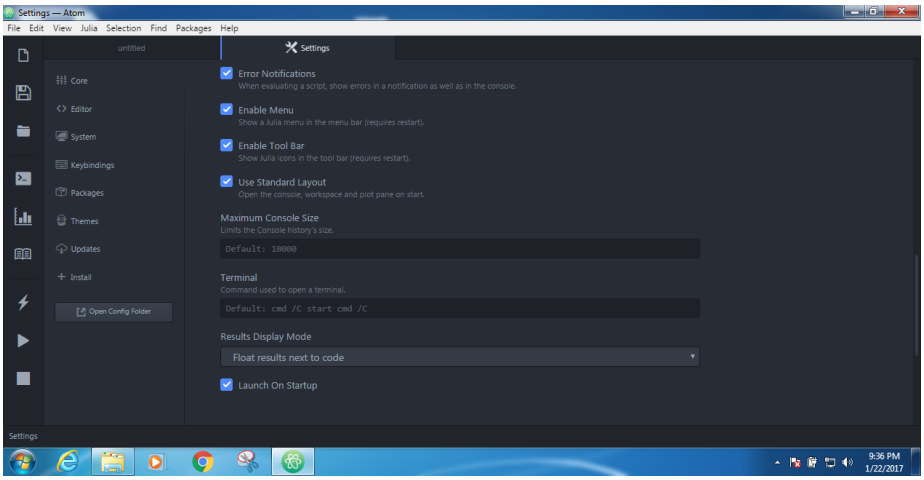
The Julia settings tab available from the Julia menu allows for updating various settings available as part of the julia-client Atom package.



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.

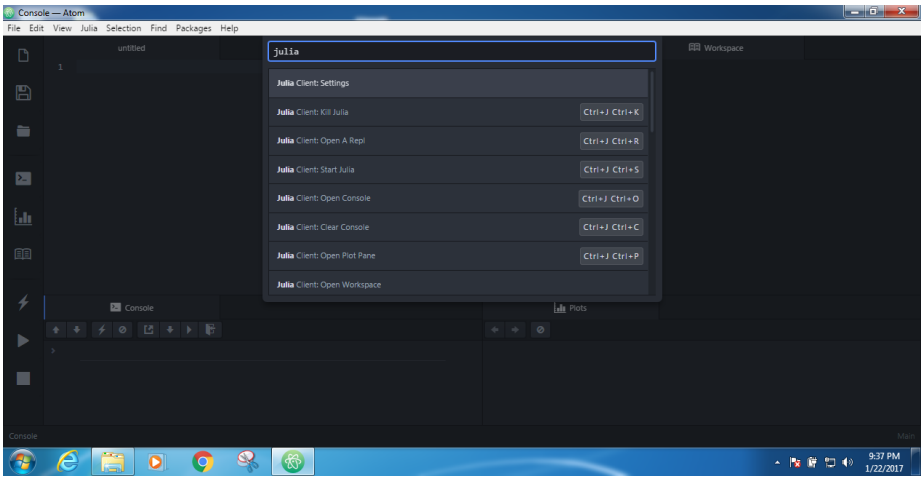




Available Julia Options include: * Boot Mode – options include Basic, Cycler and Server. It is not recommended to update this setting for a JuliaPro installation. * Deprecation warnings – display deprecation warnings * Optimization Level – the level of optimizations used in compiling Julia code. Higher 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.

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 Ctrl+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:

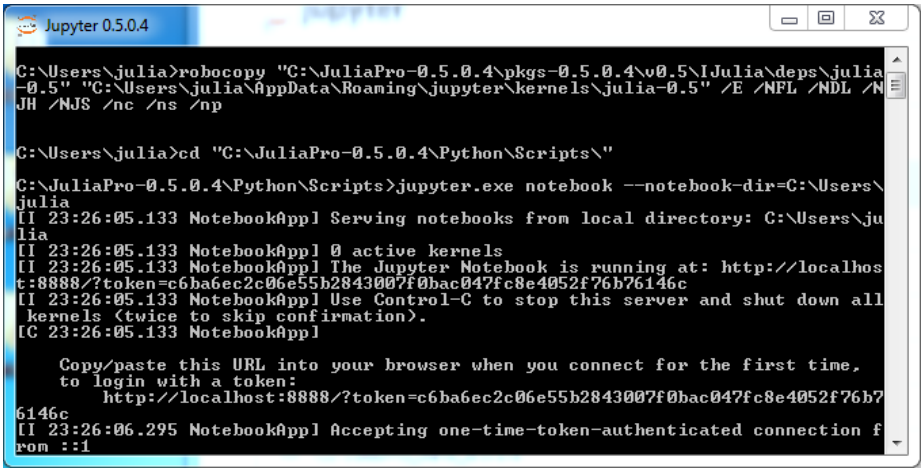
- Julia Client: Settings
- Julia Client: Run File
- Julia Client: Run Block
- Julia Client: Kill Julia (Ctrl+J Ctrl+K)
- Julia Client: Goto Symbol
- Julia Client: Open a Repl (Ctrl+J Ctrl+R)
- Julia Client: Start Julia (Ctrl+J Ctrl+S)
- Julia Client: Open Console (Ctrl+J Ctrl+O)
- Julia Client: Run and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

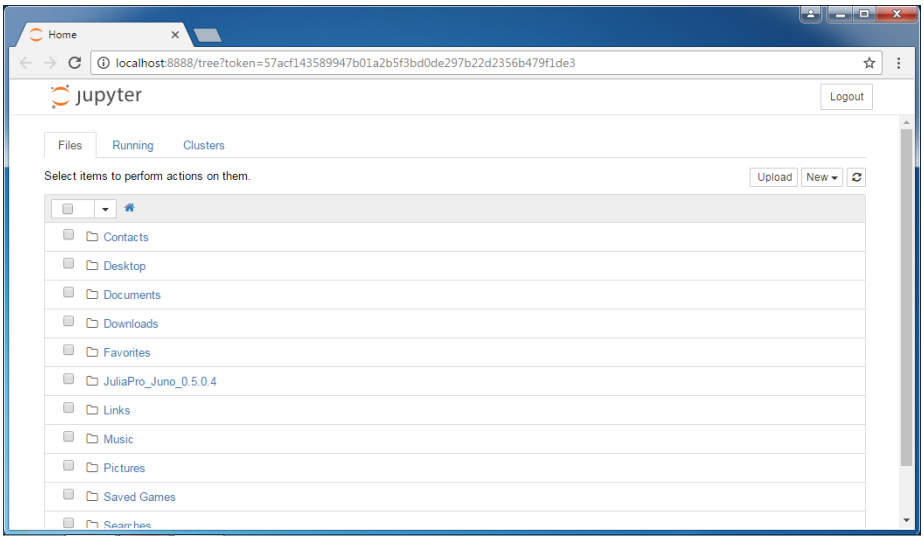
9 6. Using IJulia and Jupyter in JuliaPro

If the Desktop Shortcuts components were selected during the installation process, then you should have a Jupyter icon on your desktop. Double-Click on the icon to start a Jupyter notebook session.

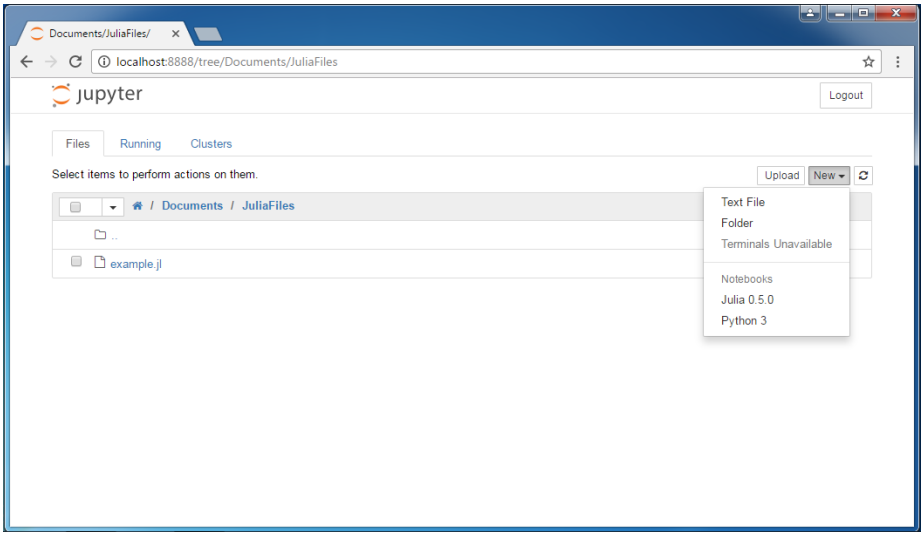
You will see a command prompt window displayed as in the following screenshot. Note the authentication token that can be used when logging-in and logging-out of the subsequent Jupyter notebook session.



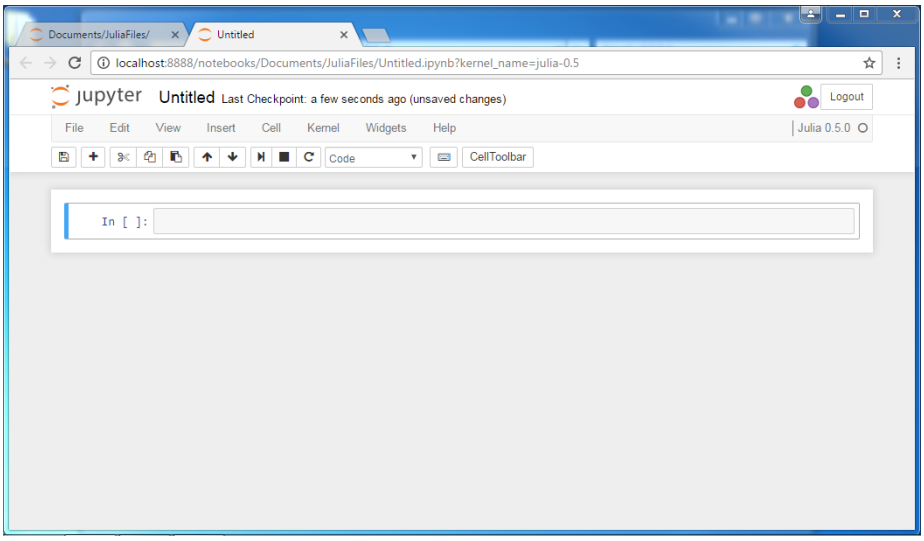
Once the Jupyter process has loaded, a Jupyter notebook server session will be displayed in your default browser window as shown below.



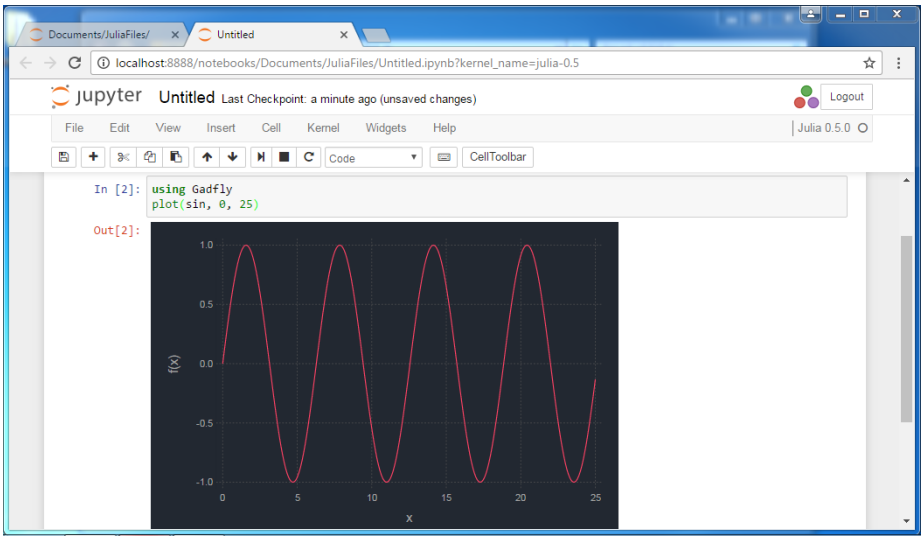
You can create a new Jupyter notebook attached to a Julia kernel by selecting a Julia 0.5.0 notebook from the "New" dropdown box on the right-hand side of the homepage of the Jupyter server session.



A new notebook running a Julia kernel with then launch as shown below.



Once the notebook has launched, you can load Julia packages, execute commands or create plots from within the notebook interface. To execute the commands within a cell, press Shift+Enter.

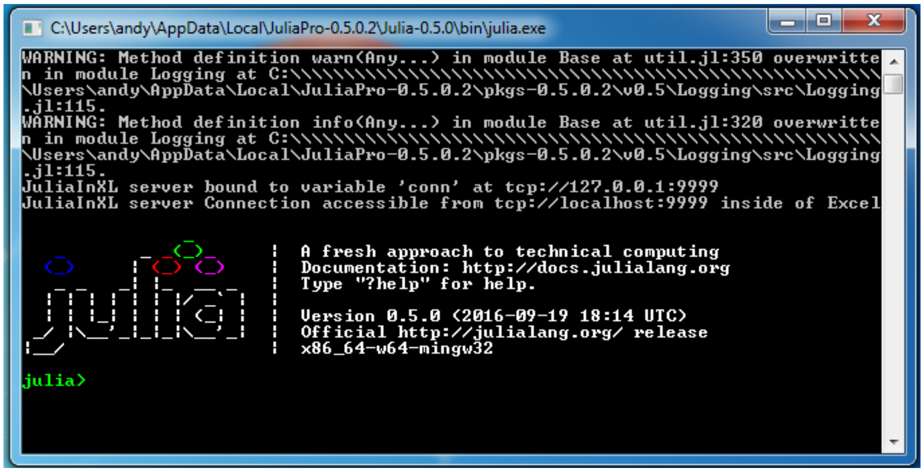


10 7. Using JuliaInXL for JuliaPro

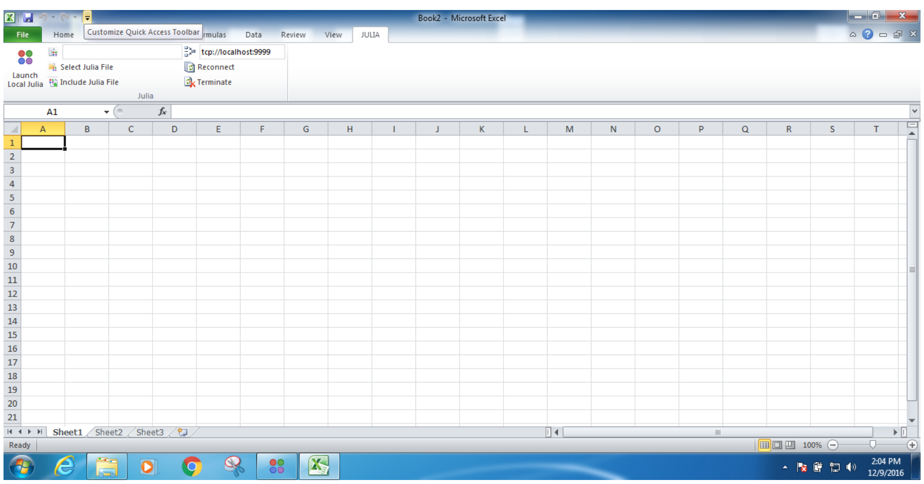
10.1 7.1 Julia Office Ribbon Tab

If JuliaInXL was selected as a component to install with your JuliaPro installation, then in most cases a Julia process should launch automatically when starting

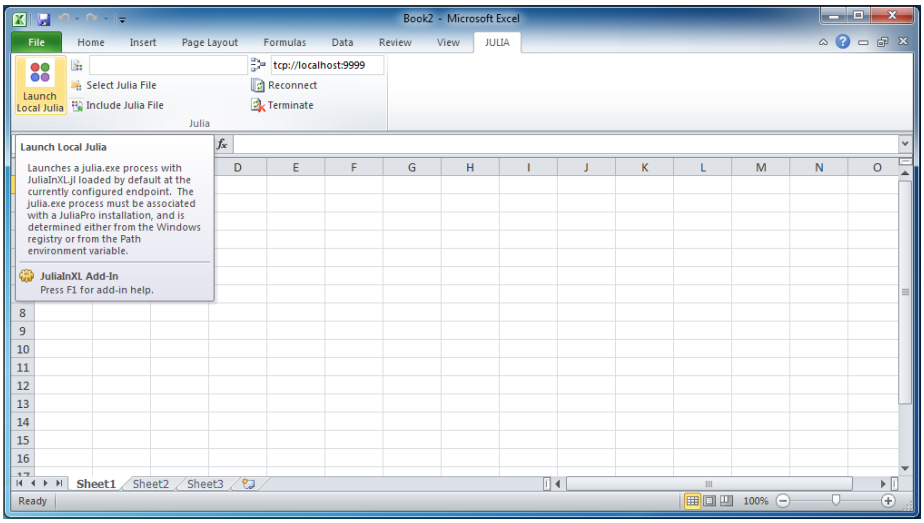
your Excel session.



A Julia tab will also be present in the Office Ribbon that contains a number of buttons and text boxes for controlling the connection between Julia and Excel, as well as loading functionality into the current Julia process.



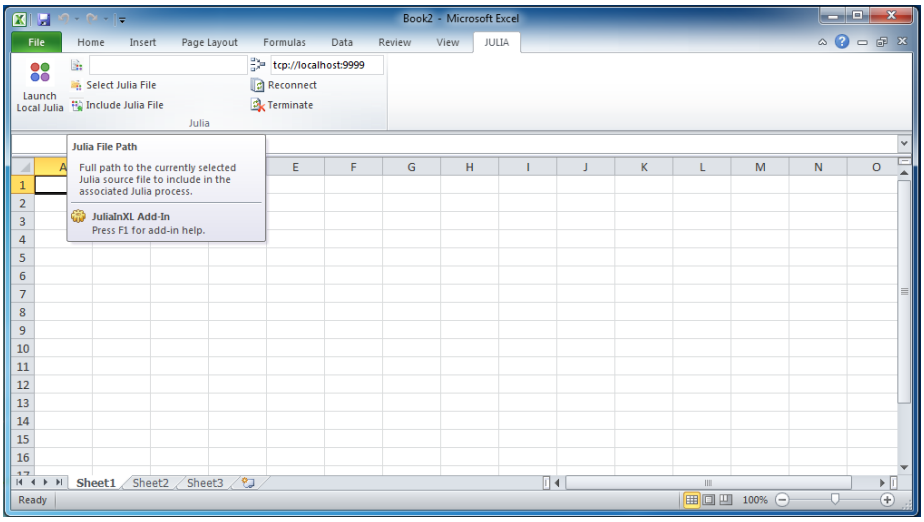
In the current version of JuliaInXL, if your Excel installation has loaded the “Analysis Toolpak - VBA” add-in, then the Julia process does not launch automatically on startup. In this scenario, you must launch the julia.exe process manually using the “Launch Local Julia” button as shown below.



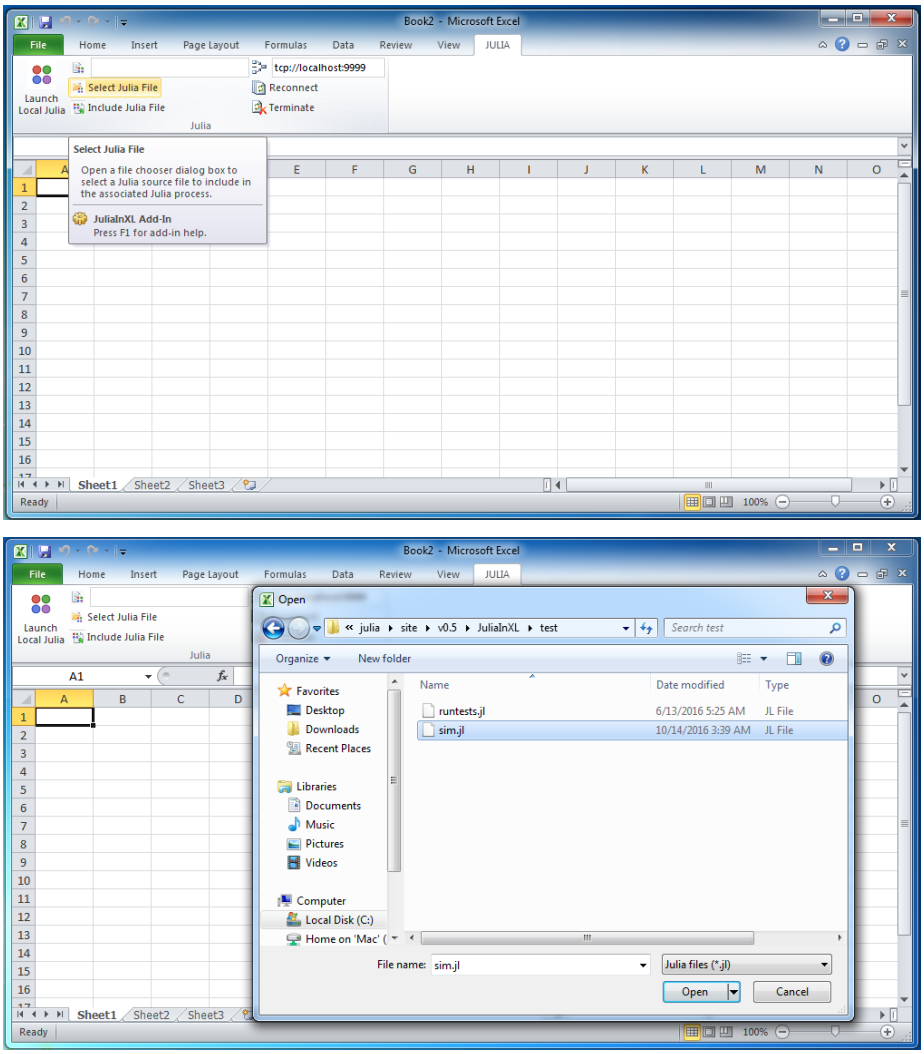
The "Launch Local Julia" button will launch a new child Julia process, as well as start a JuliaInXL server process that listens on the currently defined TCP endpoint.

When this button is pressed, any current child Julia process is shutdown before launching a new Julia process.

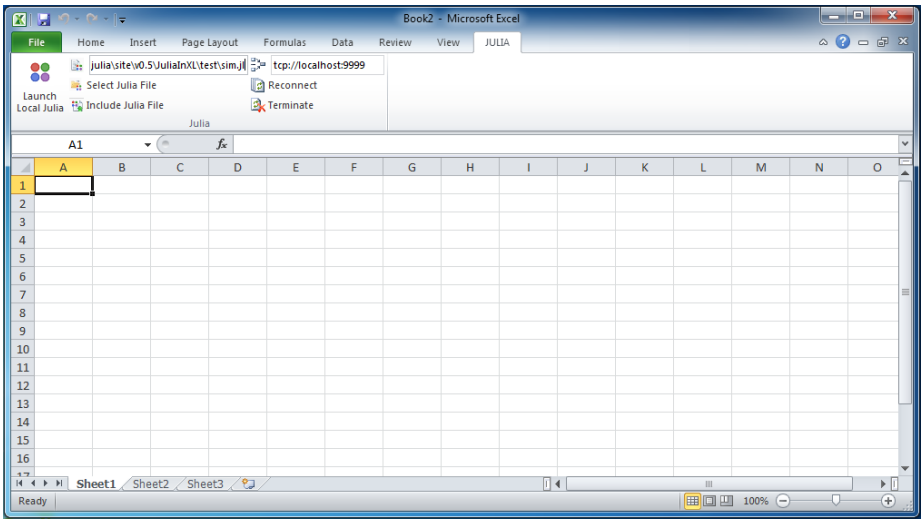
Adjacent to the "Launch Local Julia" button is a "Julia File Path" text box for entering the path to a file that can be loaded into the Julia process via the `include` command.



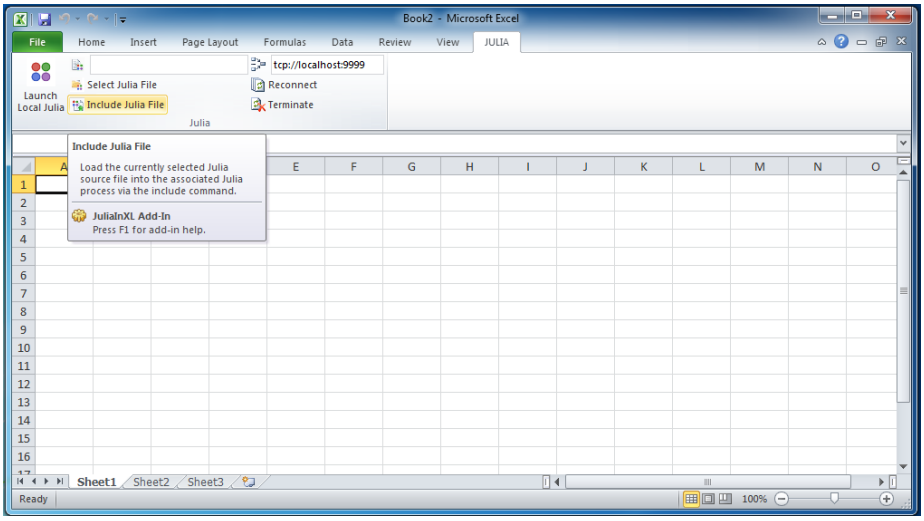
Below the "Julia File Path", is a "Select Julia File" button, which launches a file chooser dialog box that allows for browsing to a Julia file that can be loaded into the current Julia process.



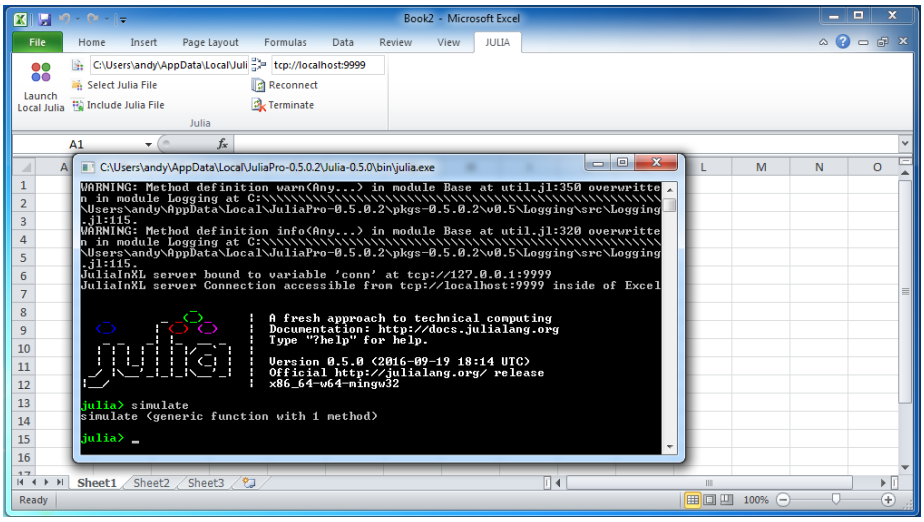
Selecting a file using this dialog box only populates the "Julia File Path" text box with the path to the file selected.



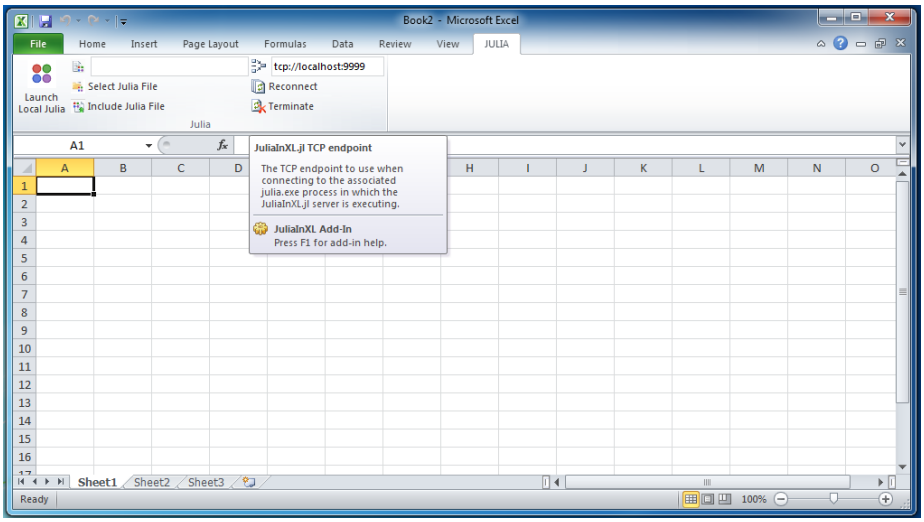
With a file selected via the "Select Julia File" button or manually entered into the "Julia File Path" text box, the selected file can be loaded into the Julia process using the "Include Julia File" button.



In the screenshot below, we have included the "sim.jl" file from the "test" directory of the JuliaInXL package installation. The `simulate` function defined in `sim.jl` is now available for use from the current Julia process and is callable from Excel via `jlcall` as described in a later section.



Adjacent to the "Julia File Path" text box is the "JuliaInXL TCP Endpoint" textbox. This textbox displays the currently configured TCP endpoint to use when Excel connects to a JuliaInXL server.



By default, the endpoint value displayed in this textbox is associated with the value stored in the "JuliaInXL_Default_Endpoint" entry of the JuliaPro Windows registry key.

For a "Current User" installation of JuliaPro, this registry key is located at "HKEY_CURRENT_USER\Software\JuliaProfessional\0.5.0.4\".

For an "All Users" installation of JuliaPro, this registry key is located at "HKEY_LOCAL_MACHINE\Software\JuliaProfessional\0.5.0.4\".

For a "Shared Drive" installation of JuliaPro, no Windows registry keys are

written on installation, but JuliaInXL will also look to see if an environment variable JULIAINXL_DEFAULT_ENDPOINT has been set.

As shown below, for "Shared Drive" installations, you should both set a value for JULIAINXL_DEFAULT_ENDPOINT, and also ensure that the path to the julia.exe executable included in your JuliaPro installation is included in a Path environment variable for either your system or your current user account.

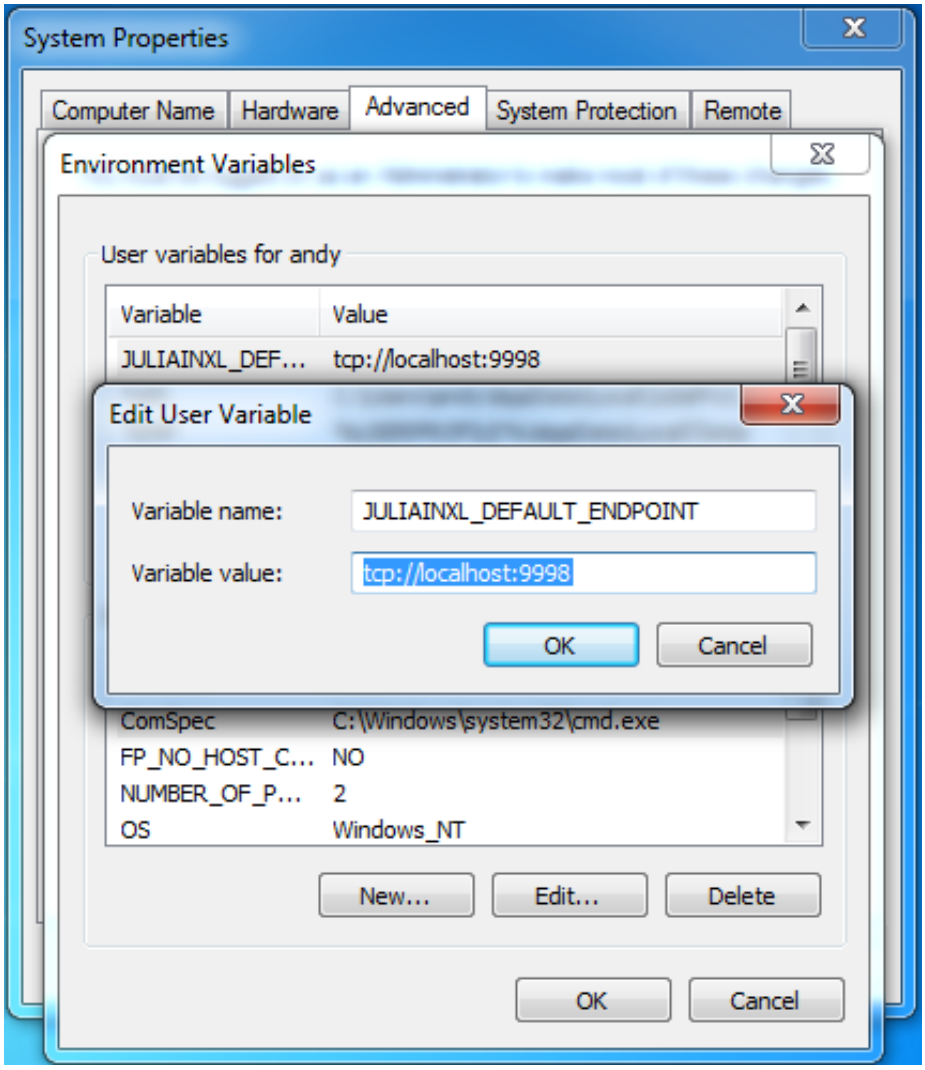


Figure 1:

For connections made to Julia processes executing on the local machine, the hostname included in the provided TCP endpoint should always be "localhost".

On the Julia side, the IP address `127.0.0.1` is used when creating the connection endpoint from which the JuliaInXL server can accept connections. Connections endpoints entered into the "JuliaInXL TCP Endpoint" on the Excel side should use the DNS name associated with an IP address, while on the Julia side the IP address should be used directly.

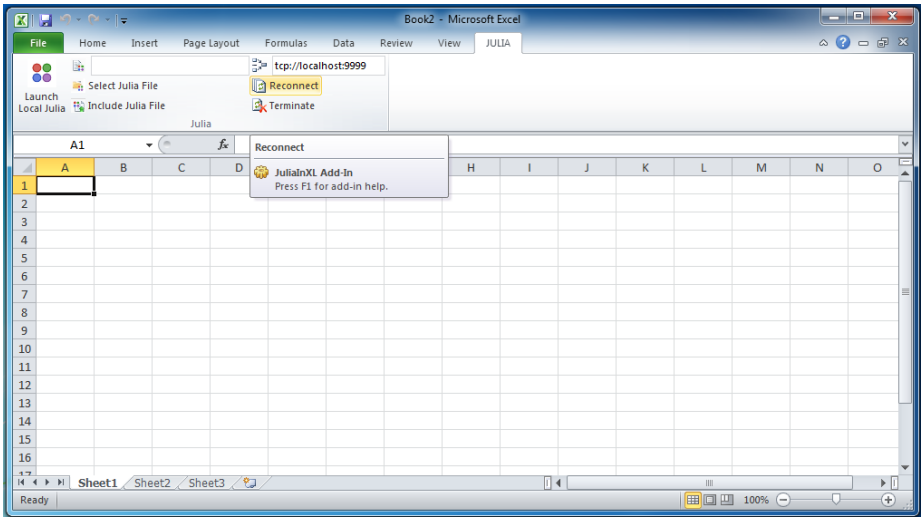
If you wish to configure your JuliaInXL session to connect to a particular JuliaInXL server, possibly on a different machine, then the value of the current endpoint can be changed either manually in the "JuliaInXL TCP Endpoint" text box, through the Windows Registry or via an environment variable.

Using the Windows Registry or an Environment variable allows for the possibility of connecting to a remote JuliaInXL server session as part of an automated workflow that launches Excel and makes use of Julia.

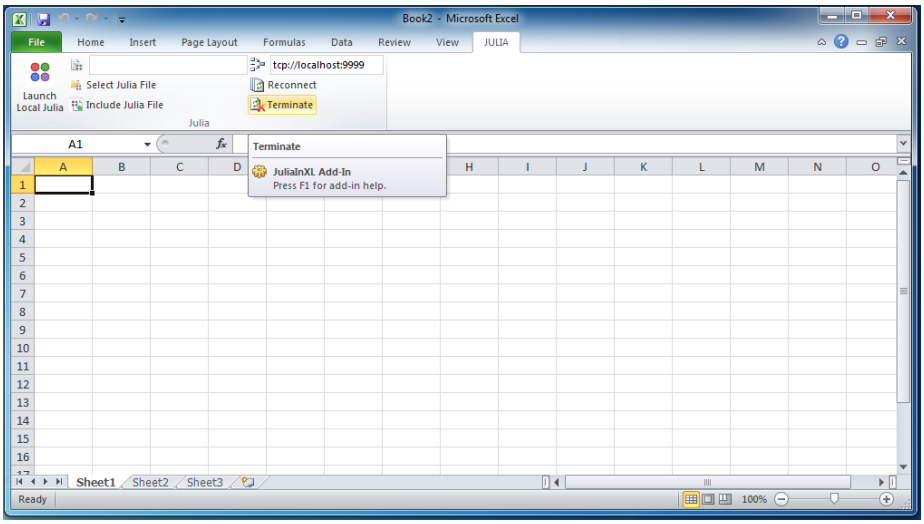
Note that with the current version of JuliaInXL, if the Excel installation has loaded the "Analysis Toolpak - VBA" Add-In, then JuliaInXL cannot be used in the automated workflow described above.

Also note that regardless of the endpoint value (e.g. tcp://hostname:) provided within the Windows Registry, in a JULIAINXL_DEFAULT_ENDPOINT environment variable, entered manually in the "JuliaInXL TCP Endpoint" text box, if a user presses the "Launch Local Julia" button, then the endpoint value in the "JuliaInXL TCP Endpoint" text box will be updated to point to "tcp://localhost:" before launching a new Julia process to create a JuliaInXL server.

Below the "JuliaInXL TCP Endpoint" textbox is the "Reconnect" button. This button resets the TCP client endpoint on the Excel side of the connection, and then attempts to reconnect to the existing JuliaInXL server.



The "Terminate" button disconnects the TCP client endpoint on the Excel side of the connection.

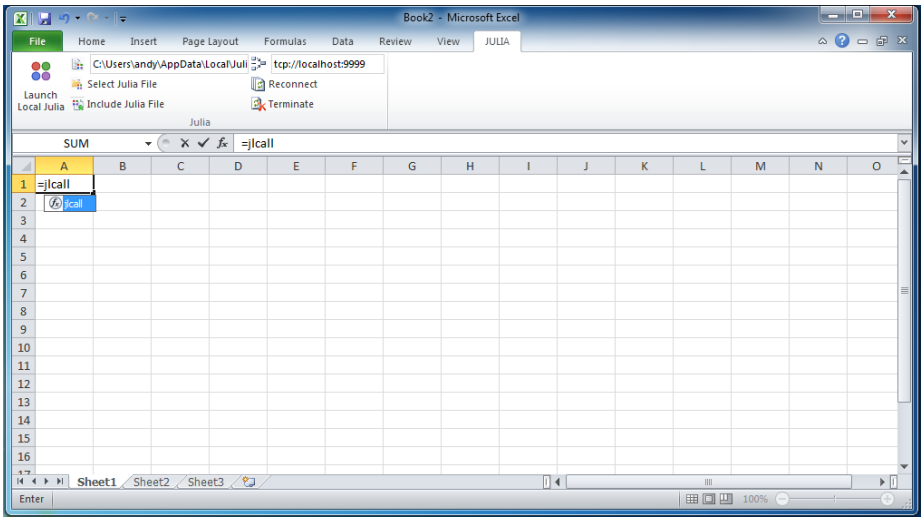


10.2 7.2. Calling Julia Functions from Excel using `jlcall`

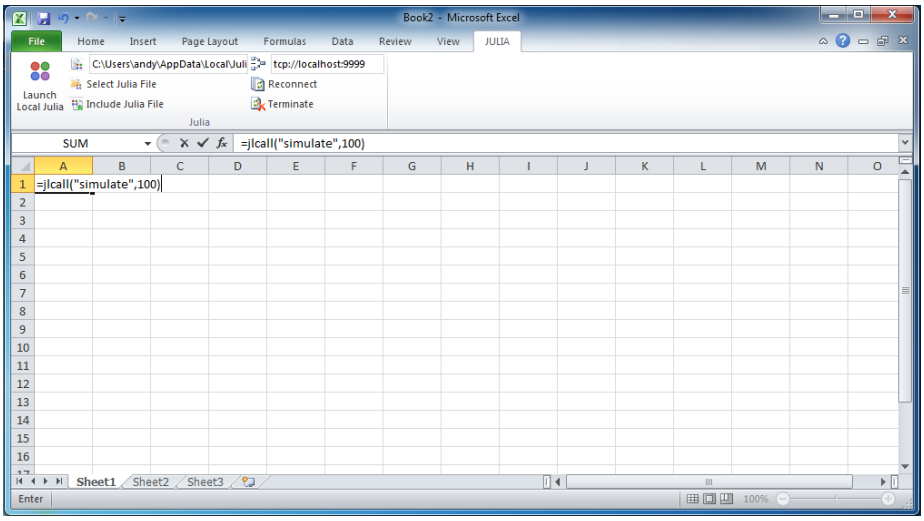
Once the server is started, julia functions can be called from Excel using the `jlcall` worksheet function. The first argument to `jlcall` is a string, which is the name of the registered Julia function to be called. Subsequent arguments to the `jlcall` function are passed as parameters to the Julia function being called. These can be constant literals, or cell references. Arrays can be passed via cell ranges.

If the Julia function returns an array (1d or 2d), then use `jlcall` as an Excel Array function by selecting a range before entering the function, and pressing `Shift+Ctrl+Enter` to finish. Functions exposed to Excel should take floats or strings, or their arrays as arguments. In general, it is a good idea to keep the function arguments as loosely typed as possible. Therefore functions should return integers, floats, or strings; or their arrays. However, arrays of dimensions greater than two are not supported. Note that Excel stores all numbers as 64 bit IEEE floats. Therefore, be aware of the possibility of truncation if returning large, or high precision, numbers. Dates are passed in from excel as floating point numbers in its internal encoding (fractional days since 1/1/1900 or 1/1/1904). Thus, they are recieved in Julia functions as floats. They can be converted to Julia DateTime values using the xldate function.

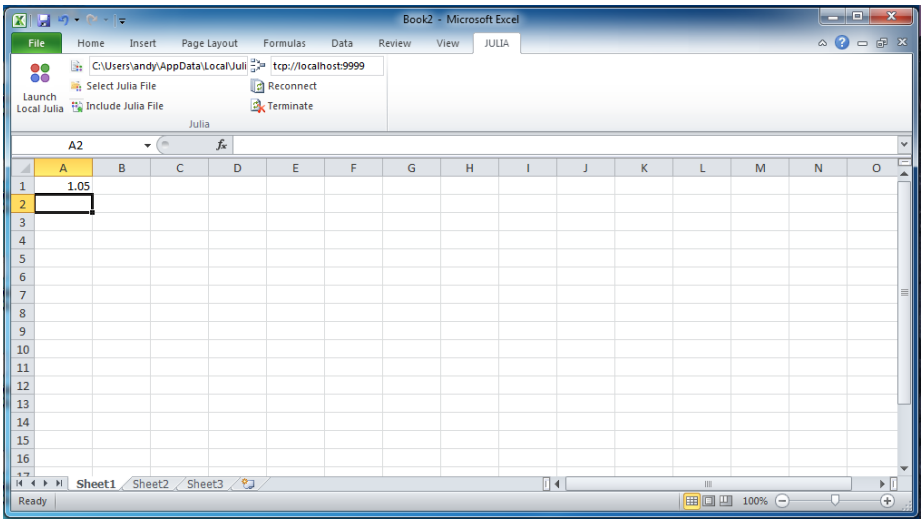
Below we show the initial entry of `jlcall` being called within a cell.



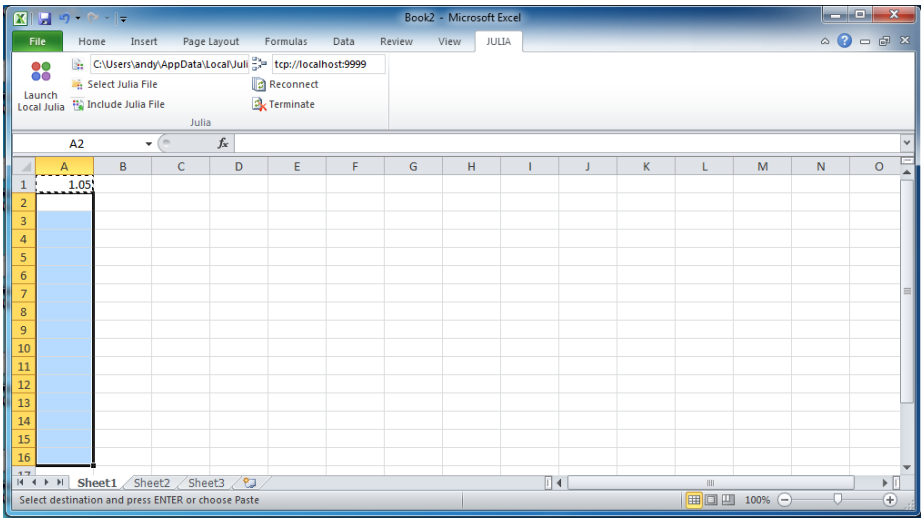
And the completion of that statement calling the `simulate` function from our example.

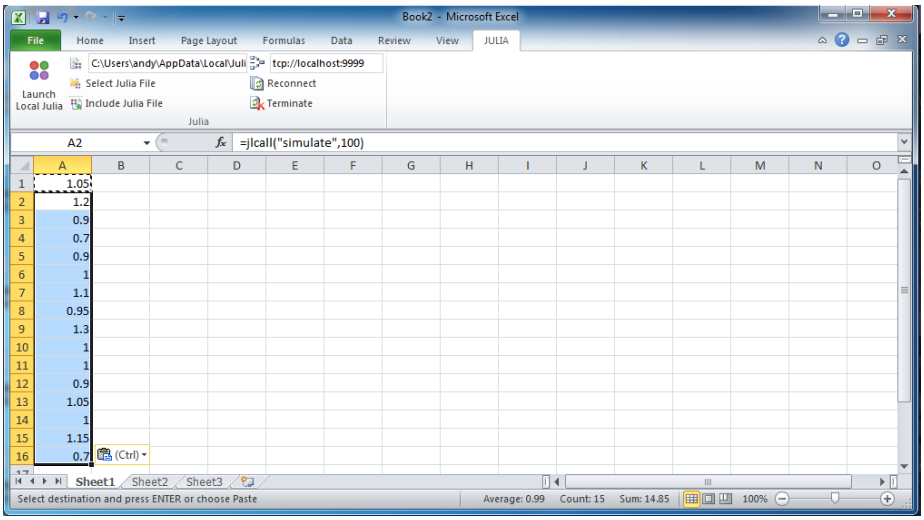


As well as the corresponding result:



By copying the contents of the cell in which `jlcall` was executed into multiple cells, the original `jlcall` operation can be repeated within multiple cells.





10.3 7.3. Defining global variables via `jlsetvar`

If you wish to assign a value to a variable within the current Julia process, a `global` variable can be created through the use of the `jlsetvar` function in Excel.

`jlsetvar` accepts two arguments, where the first argument is a text string for the name of the variable to be created and the second argument is a numeric value, a string value, or a cell reference or cell range whose contents are numbers or strings.

10.4 7.4. Executing a Julia expression via `jleval`

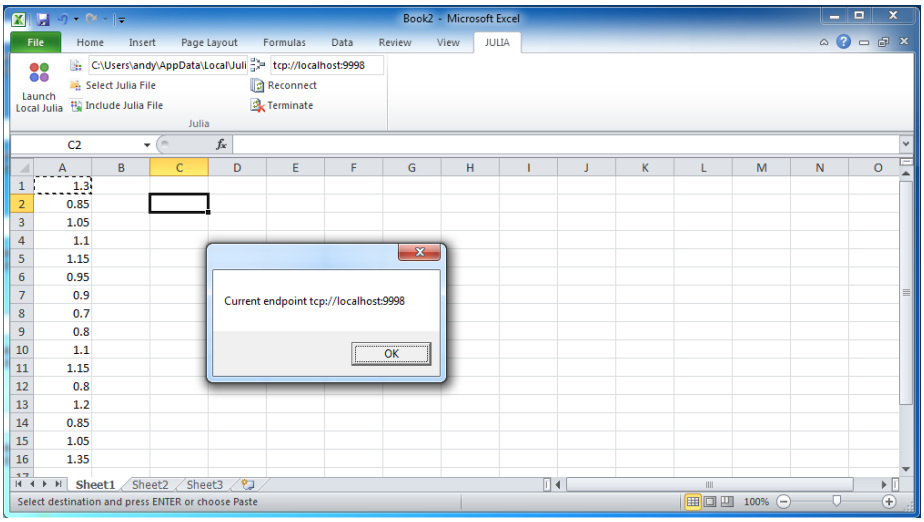
If you wish to define a Julia expression to be evaluated in the Julia process hosting the JuliaInXL server, the `jleval` function accepts a single string argument whose contents must be able to be evaluated by the julia function:

```
parse_and_eval(arg) = eval(parse(arg::String))
```

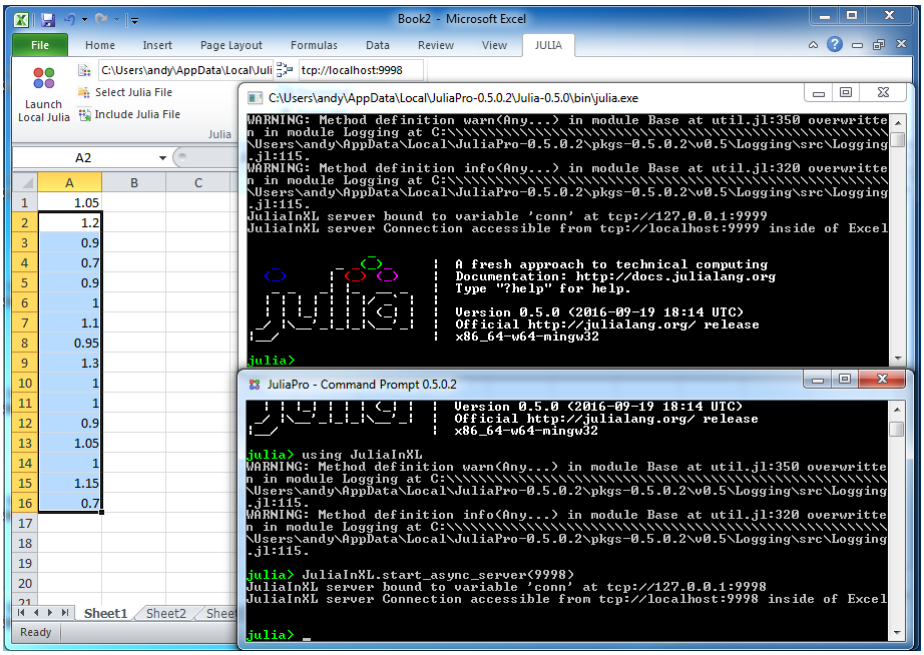
10.5 7.5. Connecting to a separate JuliaInXL server

As mentioned previously, one can connect a single Excel session to different Julia sessions by changing the port number within the Julia Office Ribbon tab.

Below is an example of changing the port number of the current Julia session from the default 9999 port to 9998.



And then connecting that same Excel session to a separate Julia session where a different connection object has been associated with the new port number.



11 8. Trademark Usage

Microsoft®, Windows®, Office® and Excel® are registered trademarks of Microsoft Corporation.

Other names may be trademarks of their respective owners.