

Introducing RStudio

RStudio™ is a new integrated development environment (IDE) for R. RStudio combines an intuitive user interface with powerful coding tools to help you get the most out of R.



Productive

RStudio brings together everything you need to be productive with R in a single, customizable environment. Its intuitive interface and [powerful coding tools](#) help you get work done faster.

Runs Everywhere

RStudio is available for [all major platforms](#) including Windows, Mac OS X, and Linux. It can even run alongside R on a server, enabling multiple users to access the RStudio IDE using a web browser.

Free & Open

Like R, RStudio is available under a [free software license](#) that guarantees the freedom to share and change the software, and to make sure it remains free software for all its users.



News

RStudio v0.93 Available (4/11/2011)

RStudio v0.93 is now available. This release includes source editor enhancements, options for customizing pane layout and appearance, an interactive plotting package, improved handling of Unicode characters, and many more small enhancements and bug fixes. All of the details on the new release can be found in our [release notes](#)

Announcing RStudio (2/28/2011)

We are pleased to announce availability of the beta version of RStudio! RStudio is a new open-source IDE for R that runs either on your desktop or on a server (where it is accessed using a browser). We invite everyone to check out the [screenshots](#) for more details; [download](#) the product to try it out, and follow its ongoing [development](#) on github.

© 2011 RStudio, Inc.

Working in the Console

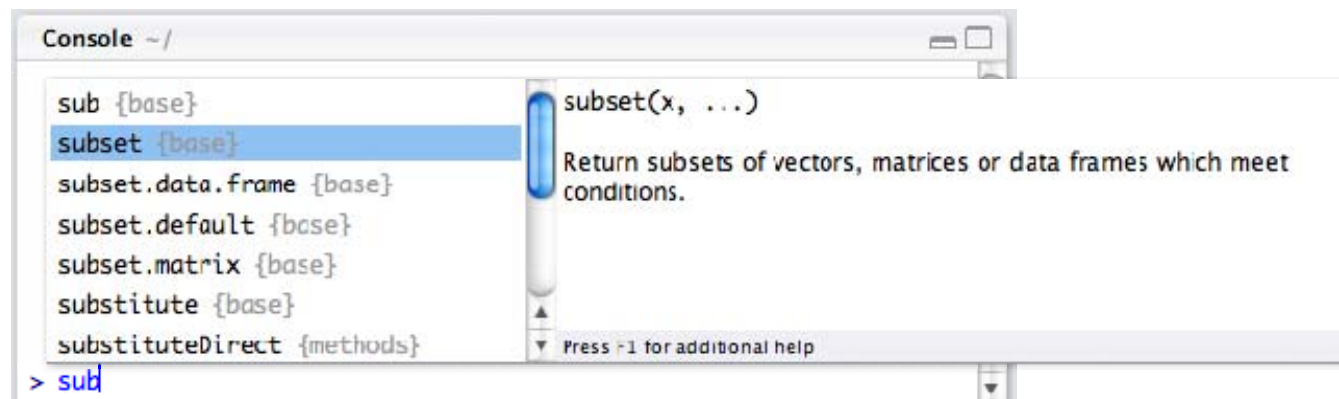
Overview

The RStudio console includes a variety of features intended to make working with R more productive and straightforward. This article reviews these features. Learning to use these features along with the related features available in the [Source](#) and [History](#) panes can have a substantial payoff in your overall productivity with R.

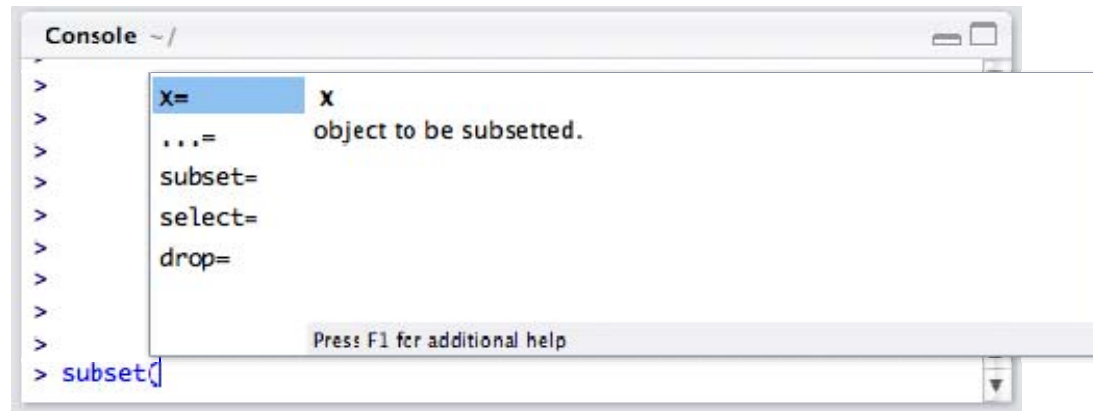
Code Completion

RStudio supports the automatic completion of code using the **Tab** key. For example, if you have an object named `pollResults` in your workspace you can type `poll` and then **Tab** and RStudio will automatically complete the full name of the object.

The code completion feature also provides inline help for functions whenever possible. For example, if you typed `sub` then pressed **Tab** you would see:



Code completion also works for function arguments, so if you typed `subset(` and then pressed **Tab** you'd see the following:

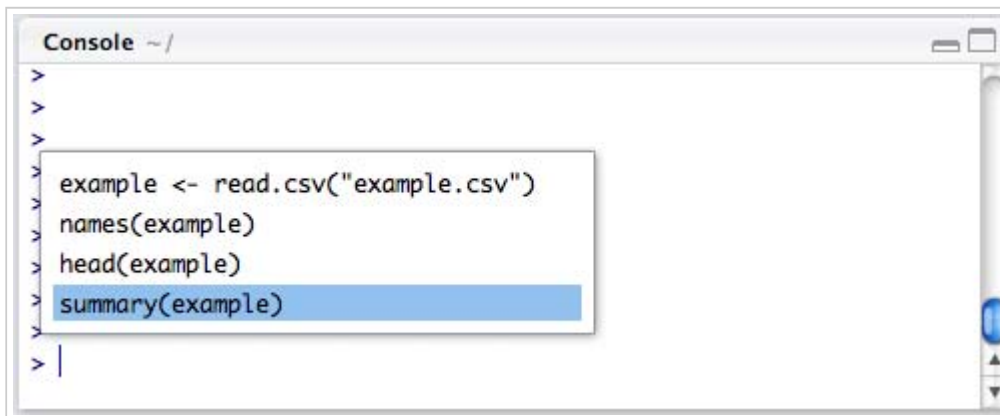


Retrieving Previous Commands

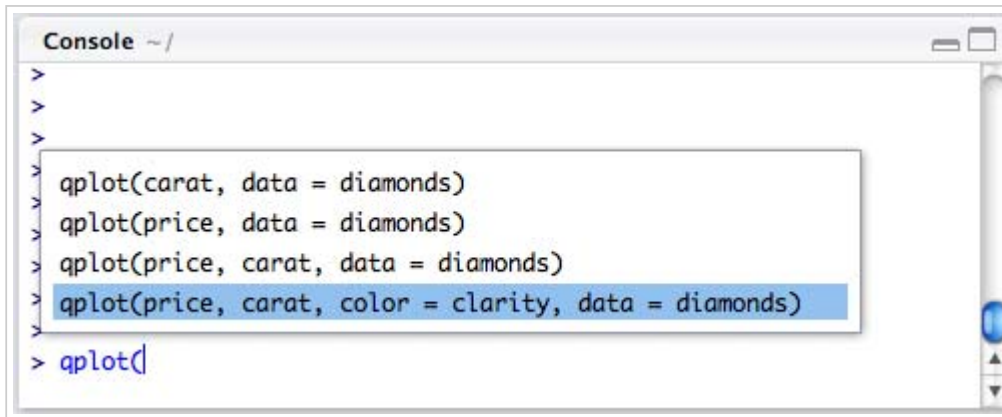
As you work with R you'll often want to re-execute a command which you previously entered. As with the standard R console, the RStudio console supports the ability to recall previous commands using the arrow keys:

- **Up** — Recall previous command(s)
- **Down** — Reverse of Up

If you wish to review a list of your recent commands and then select a command from this list you can use **Ctrl+Up** to review the list (note that on the Mac you can also use **Command-Up**):



You can also use this same keyboard shortcut to quickly search for commands that match a given prefix. For example, to search for previous instances of the `plot` function simply type `plot` and then **Ctrl+Up**:



A screenshot of an R console window titled "Console ~/". The window contains a list of commands entered in the console, with the last command highlighted in blue. The commands are:

```
>  
>  
>  
> qplot(carat, data = diamonds)  
> qplot(price, data = diamonds)  
> qplot(price, carat, data = diamonds)  
> qplot(price, carat, color = clarity, data = diamonds)  
> qplot(|
```

Console Title Bar

This screenshot illustrates a few additional capabilities provided by the Console title bar:

- Display of the current working directory.
- The ability to interrupt R during a long computation.
- Minimizing and maximizing the Console in relation to the Source pane (using the buttons at the top-right or by double-clicking the title bar).



Keyboard Shortcuts

Beyond the history and code-completion oriented keyboard shortcuts described above, there are a wide variety of other shortcuts available. Some of the more useful shortcuts include:

- **Ctrl+1** — Move focus to the Source Editor
- **Ctrl+2** — Move focus to the Console
- **Ctrl+L** — Clear the Console
- **Esc** — Interrupt R

You can find a list of all shortcuts in the [Keyboard Shortcuts](#) article.

Related Topics

- [Editing and Executing Code](#)
- [Using Command History](#)

© 2011 RStudio, Inc.

Editing and Executing Code

Overview

RStudio's source editor includes a variety of productivity enhancing features including syntax highlighting, code completion, multiple-file editing, and find/replace.

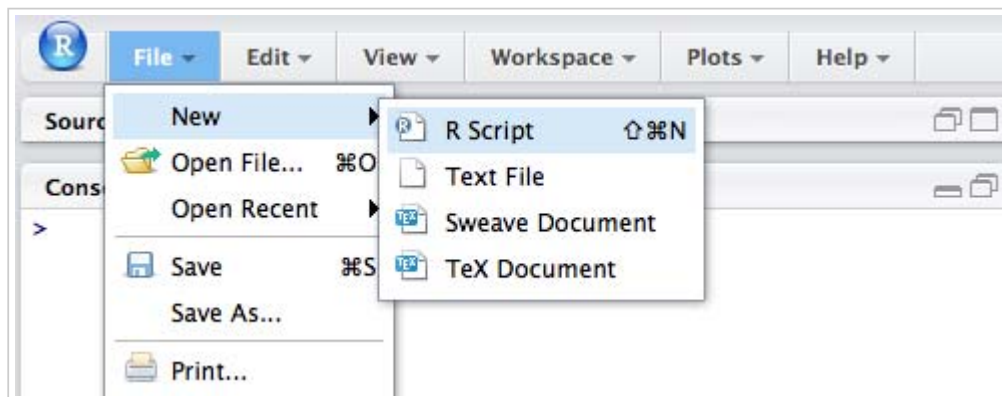
RStudio also enables you to flexibly execute R code directly from the source editor. For many R developers this represents their preferred way of working with R. By executing commands from within the source editor rather than the console it is much easier to reproduce sequences of commands as well as package them for re-use as a function. These features are described in the [Executing Code](#) section below.

Managing Files

RStudio supports syntax highlighting and other specialized code-editing features for the following types of files:

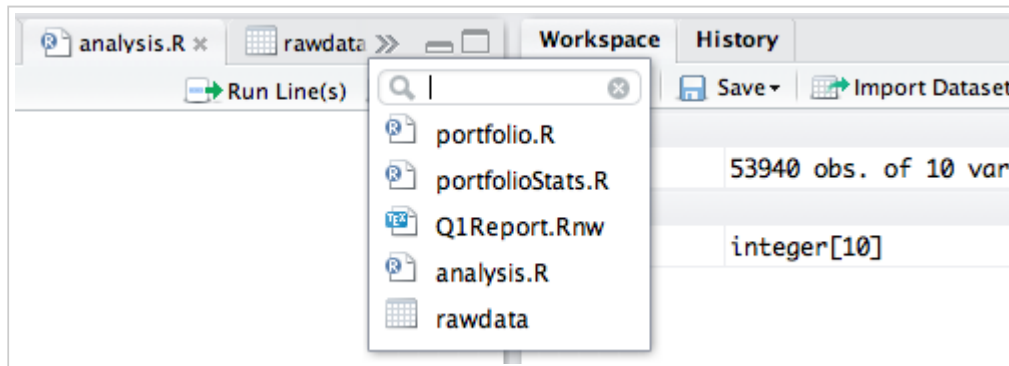
- R scripts
- Sweave documents
- TeX documents

To create a new file you use the **File -> New** menu:



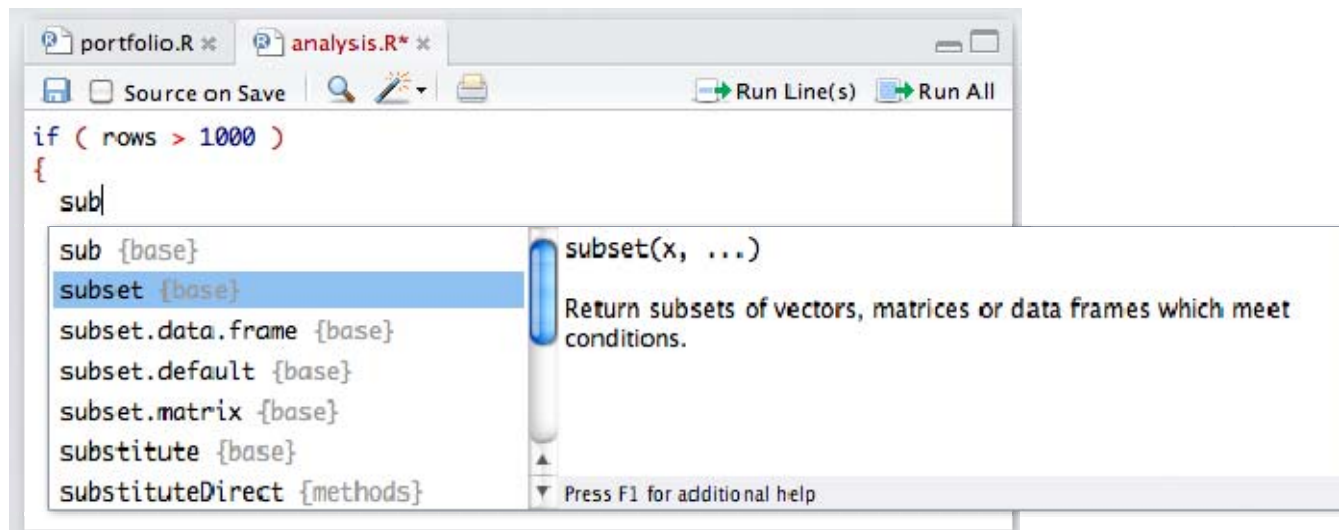
To open an existing file you use either the **File -> Open** menu or the **Open Recent** menu to select from recently opened files.

If you open several files within RStudio they are all available as tabs to facilitate quick switching between open documents. If you have a large number of open documents you can also navigate between them using the >> icon on the tab bar or the **View -> Switch to Tab** menu item:



Code Completion

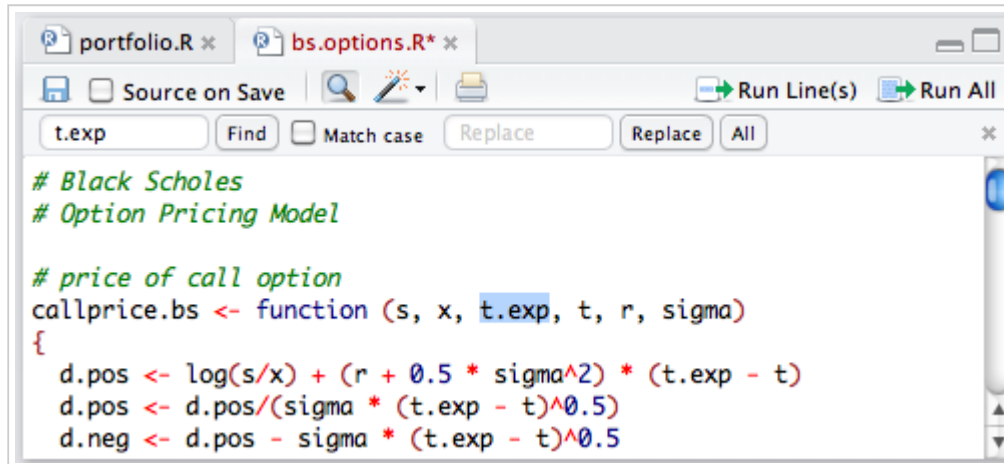
RStudio supports the automatic completion of code using the **Tab** key. For example, if you have an object named `pollResults` in your workspace you can type `poll` and then **Tab** and RStudio will automatically complete the full name of the object.



Code completion also works in the console, and more details on using it can be found the console [Code Completion](#) documentation.

Find and Replace

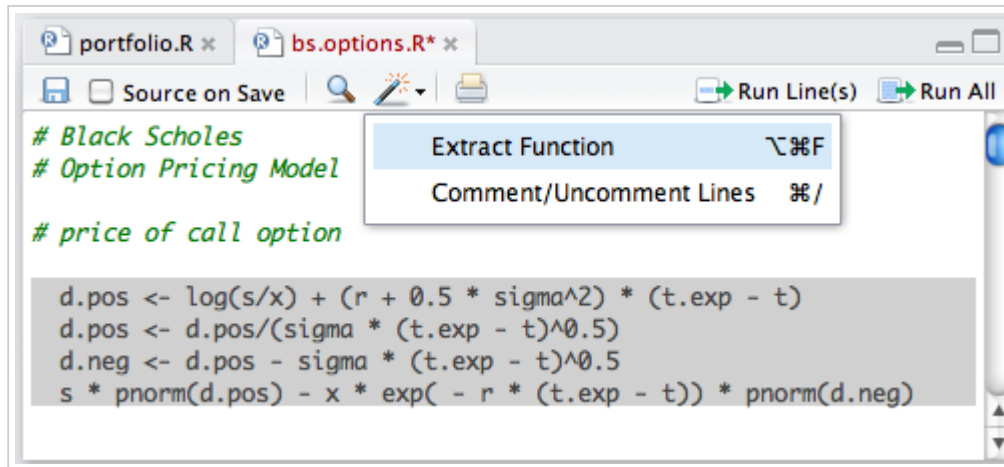
RStudio supports finding and replacing text within source documents:



Find and replace can be opened using the **Ctrl+F** shortcut key, or from the **Edit -> Find and Replace** menu item.

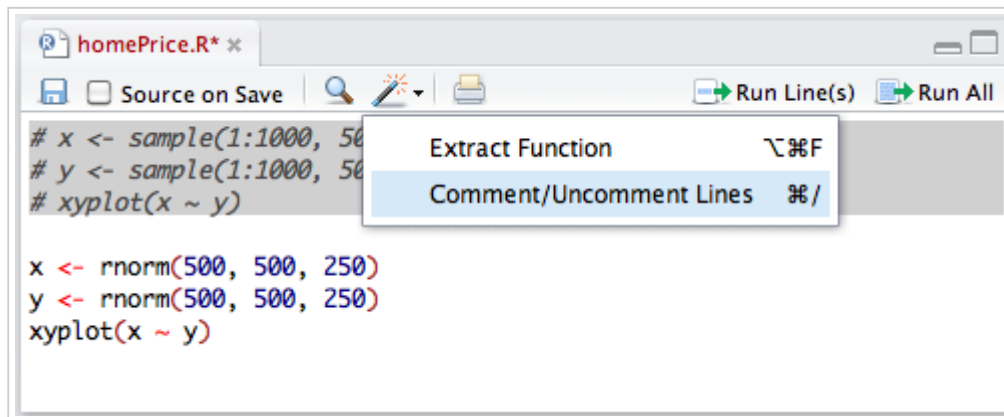
Extract Function

RStudio can analyze a selection of code from within the source editor and automatically convert it into a re-usable function. Any "free" variables within the selection (objects that are referenced but not created within the selection) are converted into function arguments:



Comment/Uncomment

You can comment and uncomment entire selections of code using the **Edit -> Comment/Uncomment Lines** menu item (you can also do this using the **Ctrl+ /** keyboard shortcut):

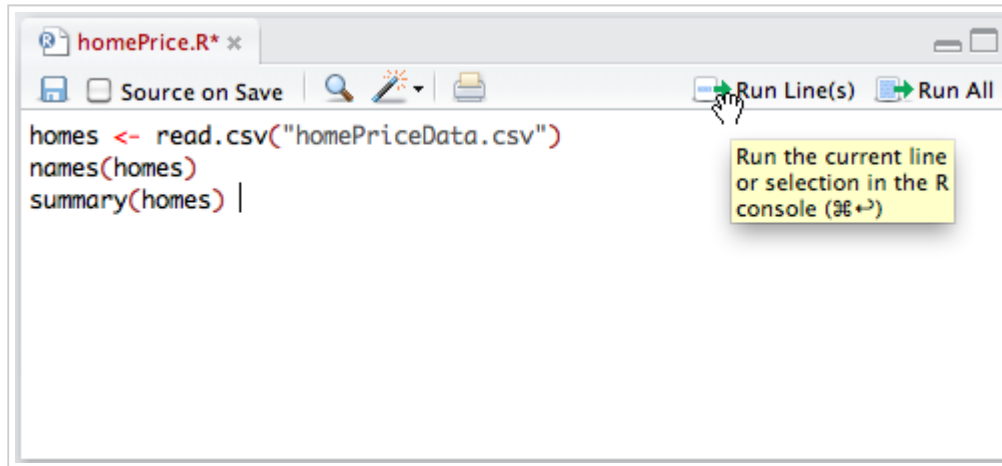


Executing Code

RStudio supports the direct execution of code from within the source editor (the executed commands are inserted into the console where their output also appears).

Executing a Single Line

To execute the line of source code where the cursor currently resides you press the **Ctrl+Enter** key (or use the **Run Line(s)** toolbar button):

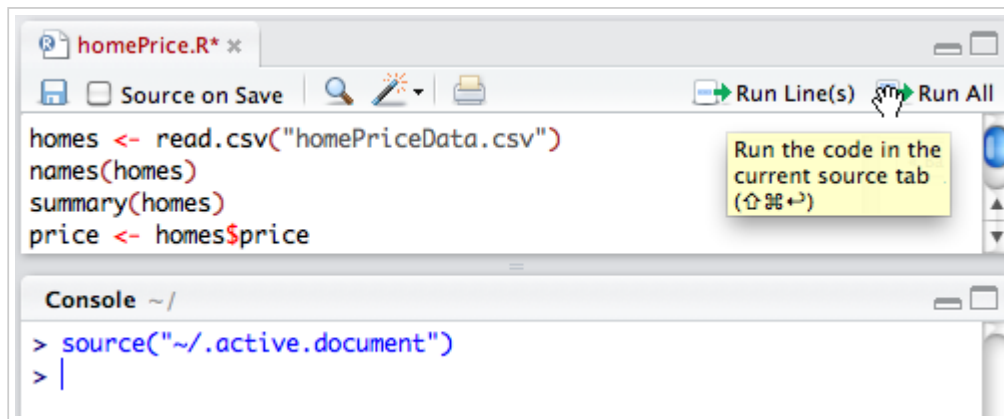


After executing the line of code, RStudio automatically advances the cursor to the next line. This enables you to single-step through a sequence of lines.

Executing Multiple Lines

There are two ways to execute multiple lines from within the editor:

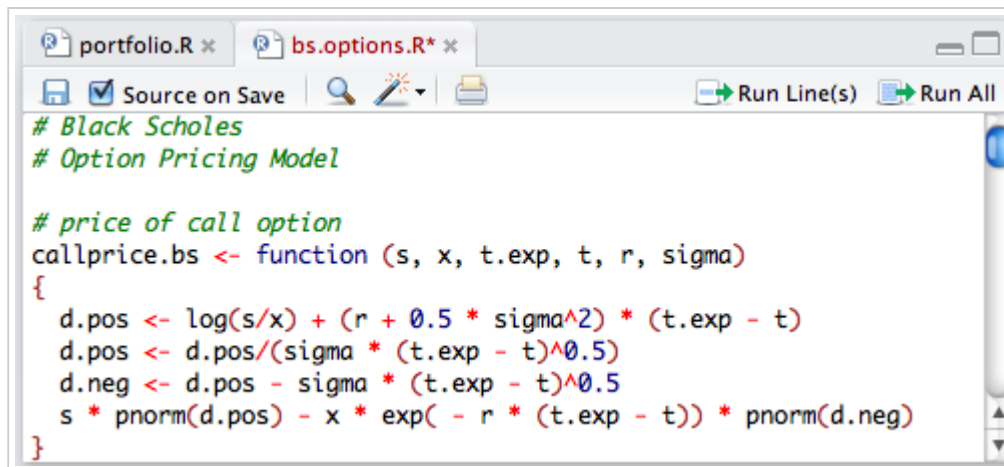
- Select the lines and press the **Ctrl+Enter** key (or use the **Run Line(s)** toolbar button); or
- To run the entire document press the **Ctrl+Shift+Enter** key (or use the **Run All** toolbar button).



The difference between running lines from a selection and invoking **Run All** is that when running a selection all lines are inserted directly into the console whereas for **Run All** the file is saved to a temporary location and then sourced into the console from there (thereby creating less clutter in the console).

Source on Save

When editing re-usable functions (as opposed to freestanding lines of R) you may wish to set the **Source on Save** option for the document (available on the toolbar next to the Save icon). Enabling this option will cause the file to automatically be sourced into the global environment every time it is saved:



Setting **Source on Save** ensures that the copy of a function within the global environment is always in sync with its source, and also provides a good way to arrange for frequent syntax checking as you develop a function.

Keyboard Shortcuts

Beyond the keyboard shortcuts described above, there are a wide variety of other shortcuts available. Some of the more useful ones include:

- **Ctrl+Shift+N** — New document
- **Ctrl+O** — Open document
- **Ctrl+S** — Save active document
- **Ctrl+I** — Move focus to the Source Editor
- **Ctrl+2** — Move focus to the Console

You can find a list of all shortcuts in the [Keyboard Shortcuts](#) article.

Related Topics

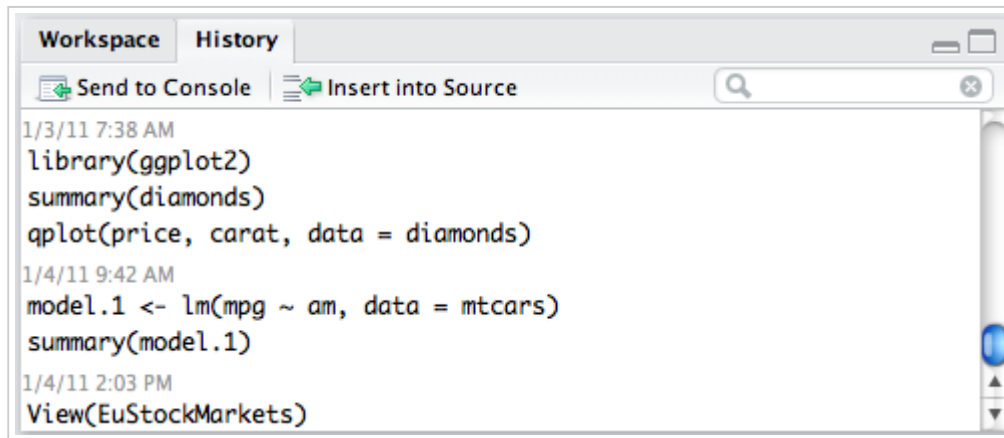
- [Working in the Console](#)
- [Using Command History](#)

Using Command History

RStudio maintains a database of all commands which you have ever entered into the Console. You can browse and search this database using the History pane.

Browsing History

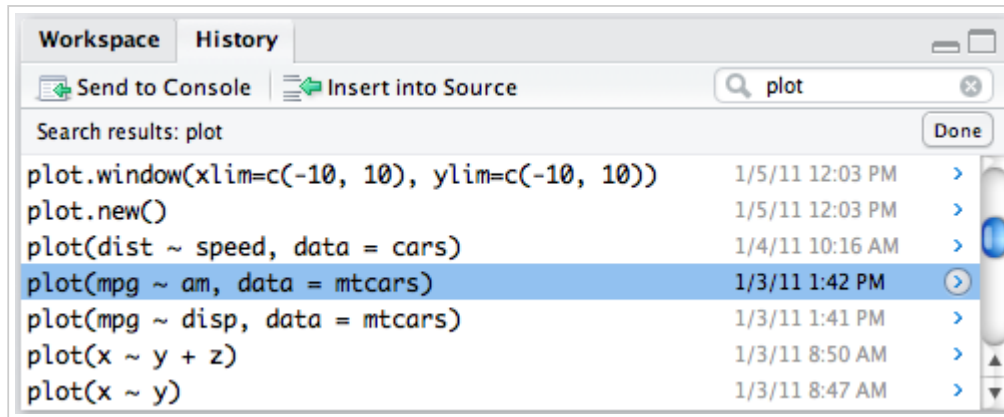
Commands you have previously entered in the RStudio console can be browsed from the History tab. The commands are displayed in order (most recent at the bottom) and grouped by block of time:



Searching History

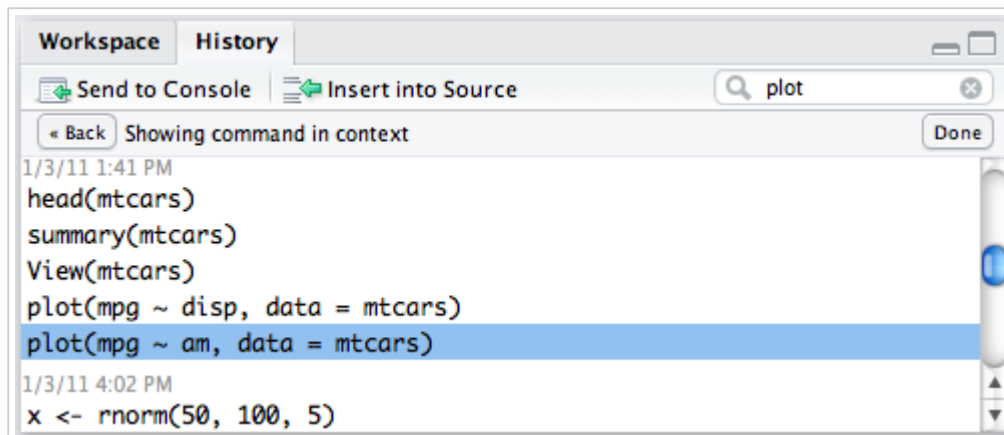
Executing a Search

You can use the search box at the top right of the history tab to search for all instances of a previous command (e.g. plot). The search can be further refined by adding additional words separated by spaces (e.g. the name of particular dataset):



Showing Command Context

After searching for a command within your history you may wish to view the other commands that were executed in proximity to it. By clicking the arrow in the right margin of the search results you can view the command within its context:



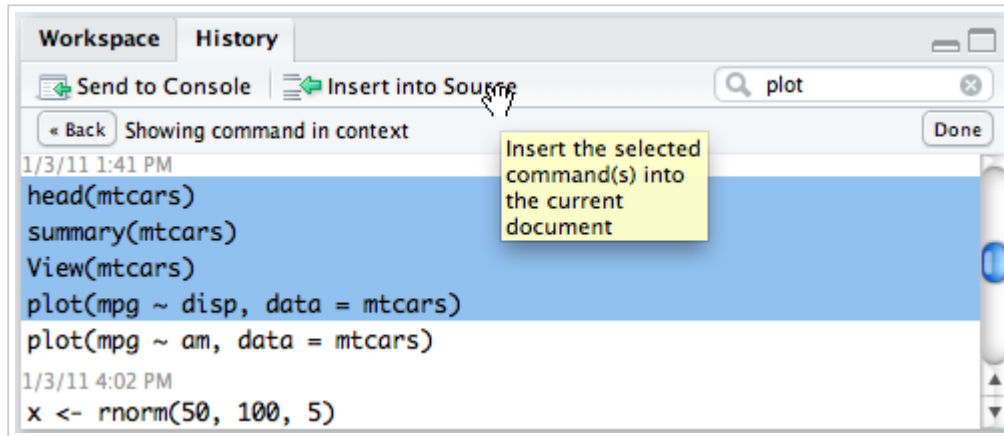
Using Commands

Commands selected within the History pane can be used in two fashions (corresponding to the two buttons on the left side of the History toolbar):

- **Send to Console**— Sends the selected command(s) to the Console. Note that the commands are inserted into the Console however they are not executed until you press **Enter**.

- **Insert into Source**— Inserts the selected command(s) into the currently active Source document. If there isn't currently a Source document available then a new untitled one will be created.

Within the history list you can select a single command or multiple commands:



Related Topics

- [Working in the Console](#)
- [Editing and Executing Code](#)

Working Directories and Workspaces

The default behavior of R for the handling of .RData files and workspaces encourages and facilitates a model of breaking work contexts into distinct working directories. This article describes the various features of RStudio which support this workflow.

Default Working Directory

As with the standard R GUI, RStudio employs the notion of a global default working directory. Normally this is the user home directory (typically referenced using ~ in R). When RStudio starts up it does the following:

- Executes the .Rprofile (if any) from the default working directory.
- Loads the .RData file (if any) from the default working directory into the workspace.
- Performs the other actions described in [R Startup](#).

When RStudio exits and there are changes to the workspace, a prompt asks whether these changes should be saved to the .RData file in the current working directory.

This default behavior can be customized in the following ways using the [RStudio Options](#) dialog:

- Change the default working directory
- Enable/disable the loading of .RData from the default working directory at startup
- Specify whether .RData is always saved, never saved, or prompted for save at exit.

Changing the Working Directory

The current working directory is displayed by RStudio within the title region of the Console. There are a number of ways to change the current working directory:

- Use the `setwd` R function
- Use the **Tools | Change Working Dir...** menu. This will also change directory location of the Files pane.
- From within the Files pane, use the **More | Set As Working Directory** menu. (Navigation within the Files pane alone will not change the working directory.)

Be careful to consider the side effects of changing your working directory:

- Relative file references in your code (for datasets, source files, etc) will become invalid when you change working directories.

- The location where .RData is saved at exit will be changed to the new directory.

Because these side effects can cause confusion and errors, it's usually best to start within the working directory associated with your project and remain there for the duration of your session. The section below describes how to set RStudio's initial working directory.

Starting in Other Working Directories

If all of the files related to a project are contained within a single directory then you'll likely want to start RStudio within that directory. There are a number of ways (which vary by platform) to do this.

File Associations

On all platforms RStudio registers itself as a handler for .RData, .R, and other R related file types. This means that the system file browser's context-menu will show RStudio as an **Open With** choice for these files.

You can also optionally create a default association between RStudio and the .RData and/or .R file types.

When launched through a file association, RStudio automatically sets the working directory to the directory of the opened file. Note that RStudio can also open files via associations when it is already running—in this case RStudio simply opens the file and does not change the working directory.

Shortcuts (Windows)

On Windows, you can create a shortcut to RStudio and customize the "Start in" field. When launched through this shortcut RStudio will startup within the specified working directory.

Drag and Drop (Mac)

On Mac, dragging and dropping a folder from the Finder on the RStudio Dock icon will cause RStudio to startup with the dropped folder as the current working directory.

Run from Terminal (Mac and Linux)

On Mac and Linux systems you can run RStudio from a terminal and specify which working directory to startup within. Additionally, on Linux systems if you run RStudio from a terminal and specify no command line argument then RStudio will startup using the current working directory of the terminal.

For example, on the Mac you could use the following commands to open RStudio (respectively) in the '~/projects/foo' directory or the current working directory:

```
$ open -a RStudio ~/projects/foo  
$ open -a RStudio .
```

On Linux you would use the following commands (note that no '.' is necessary in the second invocation):

```
$ rstudio ~/projects/foo  
$ rstudio
```

Handling of .Rprofile

When starting RStudio in an alternate working directory the .Rprofile file located within that directory is sourced. If (and only if) there is not an .Rprofile file in the alternate directory then the global default profile (e.g. ~/.Rprofile) is sourced instead.

Loading and Saving Workspaces

If you want to save or load a workspace during an RStudio session you can use the following commands to save to or load from the .RData file in the current working directory:

```
> save.image()  
> load(".RData")
```

Note that the load function appends (and overwrites) objects within the current workspace rather than replacing it entirely. Prior to loading you may therefore wish to clear all objects currently within the workspace. You can do so using the following command:

```
> rm(list=ls())
```

Note that since loading is handled at startup and saving is handled at exit, in many cases you won't require these commands. If however you change working directories within a session you may need them in order to sync your workspace with the directory you have changed to.

The RStudio **Workspace** menu also includes items that execute the above described commands, as well as enables you to load or save specific .RData files.

Handling of .Rhistory

The .Rhistory file determines which commands are available by pressing the up arrow key within the console. RStudio handles the .Rhistory file differently than the standard R GUI. This is because in addition to the .Rhistory file RStudio also includes a searchable history database (accessible via the History tab). For the sake of simplicity RStudio attempts to keep these two history contexts in sync.

The conventional handling of .Rhistory files is as follows:

- Load and save .Rhistory within the current working directory
- Only save the .Rhistory file when the user chooses to save the .RData file

Whereas the RStudio handling of .Rhistory files is:

- Load and save a single global .Rhistory file (located in the default working directory)
- Always save the .Rhistory file (even if the .RData file is not saved)

As a result, the contents of the History pane always match the up arrow history within the console.

Customizing RStudio

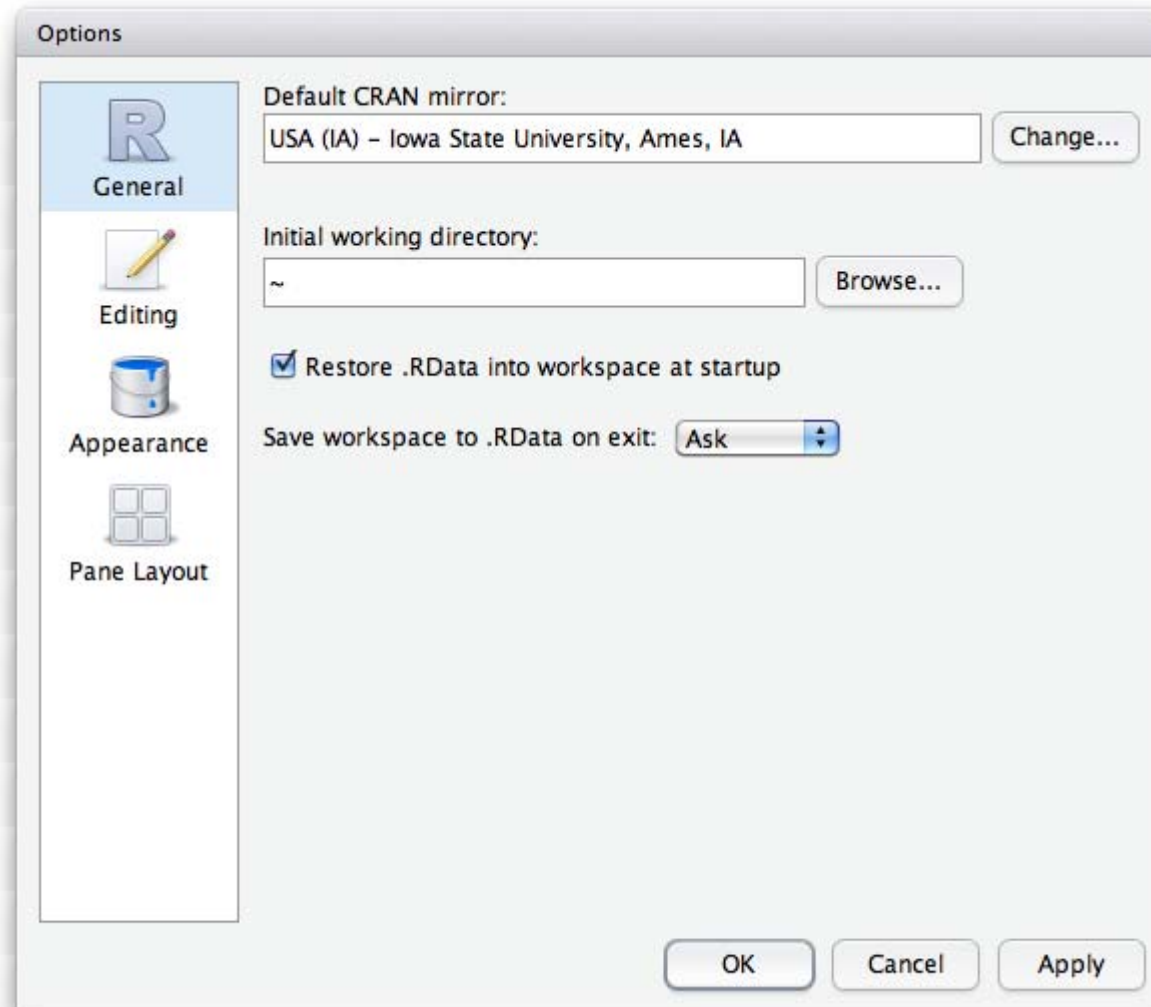
Overview

RStudio options are accessible from the Options dialog **Tools : Options** menu and include the following categories:

- [General R Options](#) — Default CRAN mirror, initial working directory, workspace save and restore behavior.
- [Source Code Editing](#) — Enable/disable line numbers, line highlighting, soft-wrapping for R files, and right margin display; configure tab spacing; set default text encoding.
- [Apperance and Themes](#) — Specify font size and visual theme for the console and source editor.
- [Pane Layout](#) — Locations of console, source editor, and tab panes; set which tabs are included in each pane.

Details on the various settings are provided in the sections below.

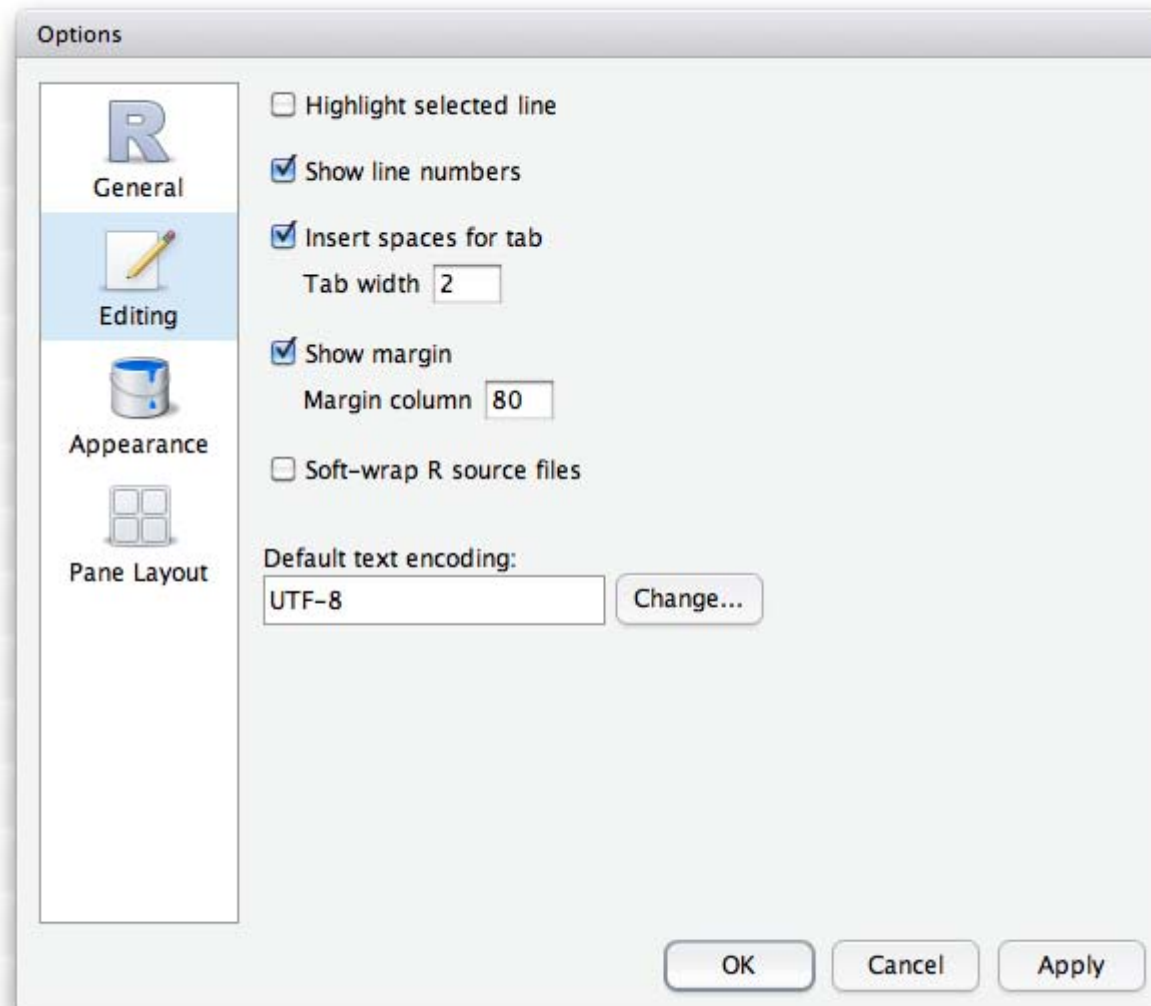
General R Options



- **Default CRAN mirror** — Set the CRAN mirror used for installing packages (can be overridden using the `repos` argument to `install.packages()`).

- **Initial working directory** — Startup directory for RStudio. The initial .RData and .Rprofile files (if any) will be read from this directory. The current working directory and Files pane will also be set to this directory. Note that this setting can be overridden when launching RStudio using a file association or a terminal with a command line parameter indicating the initial working directory.
- **Restore .RData into workspace at startup** — Load the .RData file (if any) found in the initial working directory into the R workspace (global environment) at startup. If you have a very large .RData file then unchecking this option will improve startup time considerably.
- **Save workspace to .RData on exit** — Ask whether to save .RData on exit, always save it, or never save it. Note that if the workspace is not dirty (no changes made) at the end of a session then no prompt to save occurs even if Ask is specified.

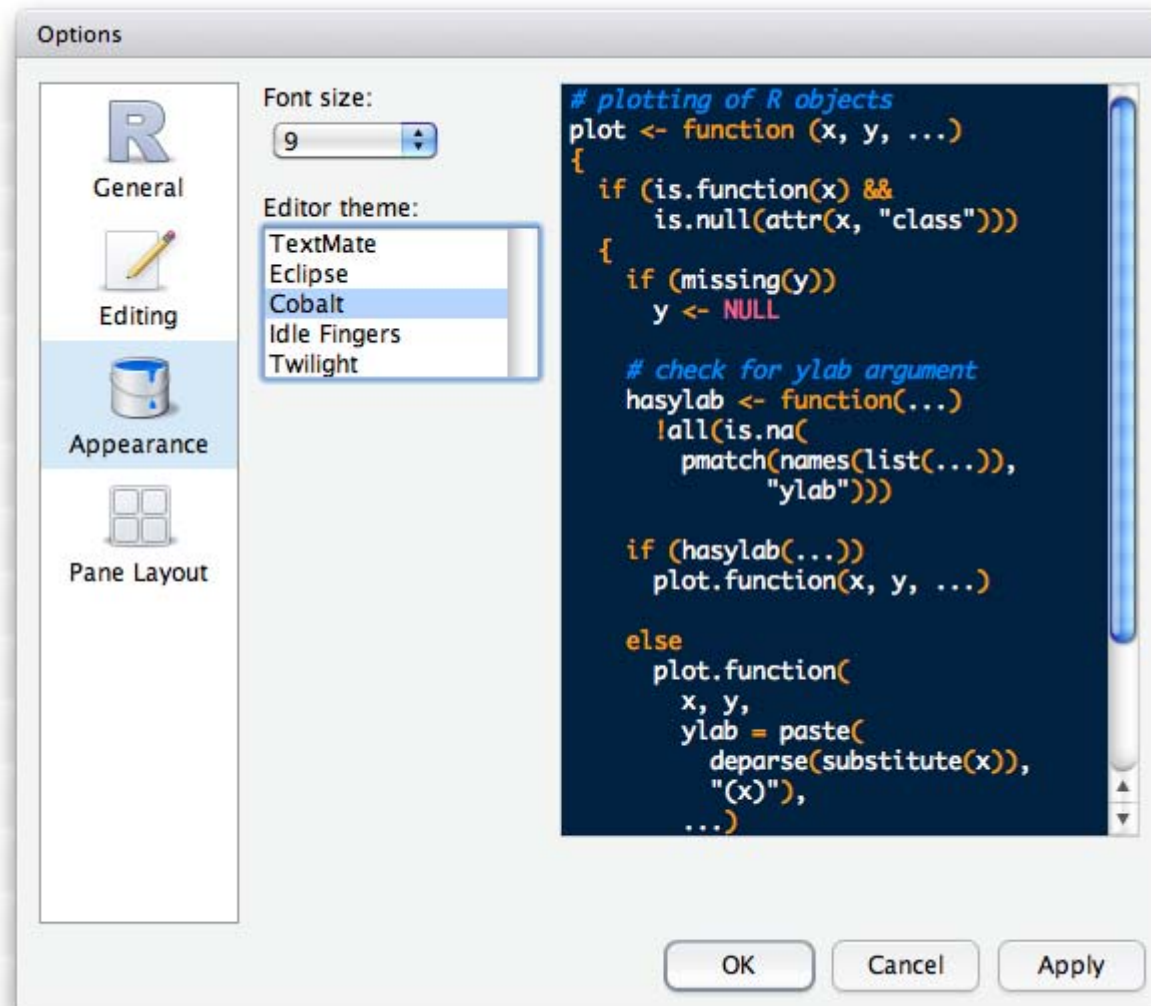
Source Code Editing



- **Highlight selected line** — Add a background highlight effect to the currently selected line of code.
- **Show line numbers** — Show or hide line numbers within the left margin.

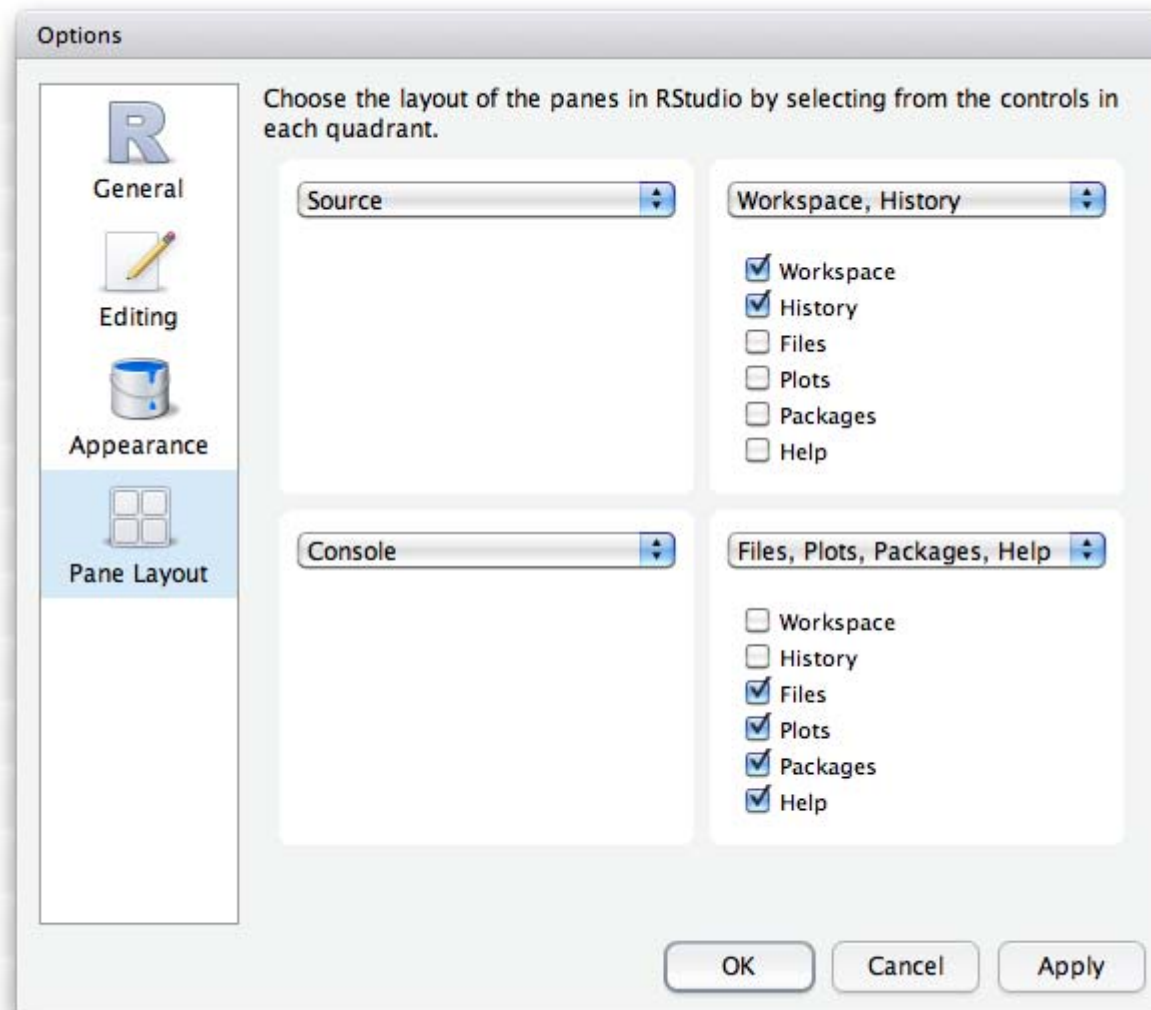
- **Insert spaces for tab** — Determine whether the tab key inserts multiple spaces rather than a tab character (soft tabs). Configure the number of spaces per soft-tab.
- **Show margin** — Display a margin guide on the right-hand side of the source editor at the specified column.
- **Soft-wrap R source files** — Wrap lines of R source code which exceed the width of the editor onto the next line. Note that this does not insert a line-break at the point of wrapping, it simply displays the code on multiple lines in the editor.
- **Default text encoding** — Specify the default text encoding for source files. Note that source files which don't match the default encoding can still be opened correctly using the **File : Reopen with Encoding** menu command.

Appearance and Themes



- **Font size** — Set the font size (in points) for panes which display code (Console, Source, History, and Workspace).
- **Editor theme** — Specify the visual theme for the Console and Source panes. You can preview the theme using the inline preview or by pressing the Apply button.

Pane Layout



- Specify the location and tab sets of panes within RStudio.
- Each of the 4 panes is always displayed (it isn't currently possible to hide a pane).

© 2011 RStudio, Inc.

Keyboard Shortcuts

Console

Description	Windows & Linux	Mac
Move cursor to Console	Ctrl+2	Ctrl+2
Clear console	Ctrl+L	Command+L
Move cursor to beginning of line	Home	Command+Left
Move cursor to end of line	End	Command+Right
Navigate command history	Up/Down	Up/Down
Popup command history	Ctrl+Up	Command+Up
Interrupt currently executing command	Esc	Esc
Yank line up to cursor	Ctrl+U	Command+U
Yank line after cursor	Ctrl+K	Command+K
Insert currently yanked text	Ctrl+Y	Command+Y
Insert assignment operator	Alt+-	Option+-

Source

Description	Windows & Linux	Mac
Move cursor to Source Editor	Ctrl+1	Ctrl+1
New document (except on Chrome/Windows)	Ctrl+Shift+N	Command+Shift+N
Open document	Ctrl+O	Command+O
Save active document	Ctrl+S	Command+S
Close active document	Ctrl+Shift+L	Command+Shift+L
Compile PDF (TeX and Sweave)	Ctrl+Shift+P	Command+Shift+P

Run current line/selection	Ctrl+Enter	Command+Enter
Run current document	Ctrl+Shift+Enter	Command+Shift+Enter
Switch to tab	Ctrl+Alt+Down	Ctrl+Option+Down
Previous tab	Ctrl+Alt+Left	Ctrl+Option+Left
Next tab	Ctrl+Alt+Right	Ctrl+Option+Right
First tab	Ctrl+Shift+Alt+Left	Ctrl+Shift+Option+Left
Last tab	Ctrl+Shift+Alt+Right	Ctrl+Shift+Option+Right
Extract function from selection	Ctrl+Shift+F	Command+Shift+F
Comment/uncomment current line/selection	Ctrl+/ 	Command+/
Insert assignment operator	Alt+-	Option+-
Transpose Letters		Ctrl+T
Jump to Word	Ctrl+Left/Right	Option+Left/Right
Jump to Start/End	Ctrl+Home/End or Ctrl+Up/Down	Command+Home/End or Command+Up/Down
Delete Line	Ctrl+D	Command+D
Move Lines Up/Down	Alt+Up/Down	Option+Up/Down
Copy Lines Up/Down	Ctrl+Alt+Up/Down	Command+Option+Up/Down
Select	Shift+[Arrow]	Shift+[Arrow]
Select Word	Ctrl+Shift+Left/Right	Option+Shift+Left/Right
Select to Line Start	Shift+Home	Command+Shift+Left or Shift+Home
Select to Line End	Shift+End	Command+Shift+Right or Shift+End
Select Page Up/Down	Shift+PageUp/PageDown	Shift+PageUp/Down
Select to Start/End	Ctrl+Shift+Home/End or Shift+Alt+Up/Down	Command+Shift+Up/Down
Delete Word Left	Ctrl+Backspace	Option+Backspace or Ctrl+Option+Backspace
Delete Word Right		Option+Delete

Delete to Line End		Ctrl+K
Delete to Line Start		Option+Backspace
Indent	Tab (at beginning of line)	Tab (at beginning of line)
Outdent	Shift+Tab	Shift+Tab

Editing (Console and Source)

Description	Windows & Linux	Mac
Undo	Ctrl+Z	Command+Z
Redo	Ctrl+Shift+Z	Command+Shift+Z
Cut	Ctrl+X	Command+X
Copy	Ctrl+C	Command+C
Paste	Ctrl+V	Command+V
Select All	Ctrl+A	Command+A

Completions (Console and Source)

Description	Windows & Linux	Mac
Attempt completion	Tab or Ctrl+Space	Tab or Command+Space
Navigate candidates	Up/Down	Up/Down
Accept selected candidate	Enter, Tab, or Right	Enter, Tab, or Right
Show help for selected candidate	F1	F1
Dismiss completion popup	Esc	Esc

Views

Description	Windows & Linux	Mac
Move cursor to Source Editor	Ctrl+1	Ctrl+1

Move cursor to Console	Ctrl+2	Ctrl+2
Show workspace	Ctrl+3	Ctrl+3
Show data	Ctrl+4	Ctrl+4
Show history	Ctrl+5	Ctrl+5
Show files	Ctrl+6	Ctrl+6
Show plots	Ctrl+7	Ctrl+7
Show packages	Ctrl+8	Ctrl+8
Show help	Ctrl+9	Ctrl+9

Plots

Description	Windows & Linux	Mac
Previous plot	Ctrl+PageUp	Command+PageUp
Next plot	Ctrl+PageDown	Command+PageDown

Interactive Plotting with Manipulate

RStudio includes a `manipulate` package that enables the addition of interactive capabilities to standard R plots. This is accomplished by binding plot inputs to custom controls rather than static hard-coded values.

Basic Usage

The `manipulate` function accepts a plotting expression and a set of controls (e.g. slider, picker, or checkbox) which are used to dynamically change values within the expression. When a value is changed using its corresponding control the expression is automatically re-executed and the plot is redrawn.

For example, to create a plot that enables manipulation of a parameter using a slider control you could use syntax like this:

```
library(manipulate)
manipulate(plot(1:x), x = slider(1, 100))
```

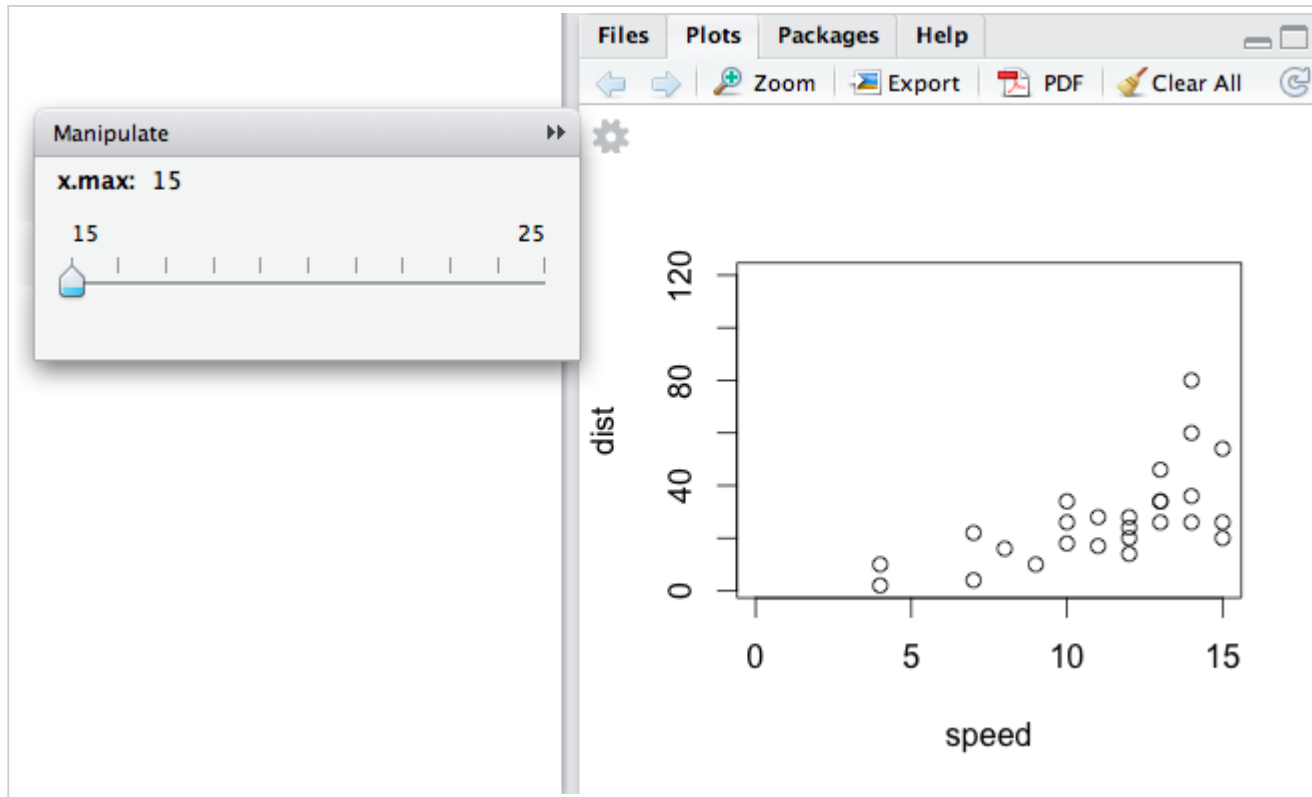
After this code is executed the plot is drawn using an initial value of 1 for `x`. A manipulator panel is also opened adjacent to the plot which contains a slider control used to change the value of `x` from 1 to 100.

Slider Control

The `slider` control enables manipulation of plot variables along a numeric range. For example:

```
manipulate(
  plot(cars, xlim=c(0,x.max)),
  x.max=slider(15,25))
```

Results in this plot and manipulator:



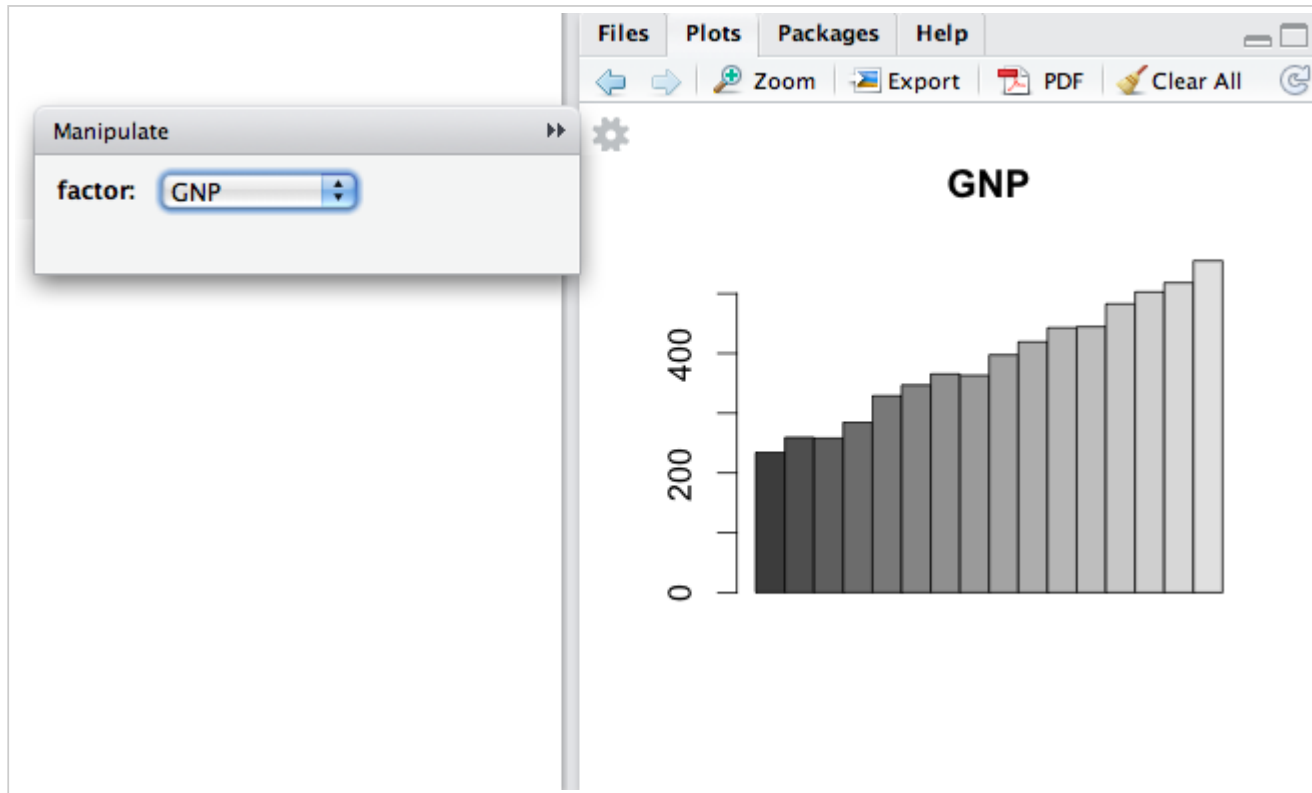
Slider controls also support custom labels and step increments.

Picker Control

The picker control enables manipulation of plot variables based on a set of fixed choices. For example:

```
manipulate(
  barplot(as.matrix(longley[,factor]),
    beside = TRUE, main = factor),
  factor = picker("GNP", "Unemployed", "Employed"))
```

Results in this plot and manipulator:



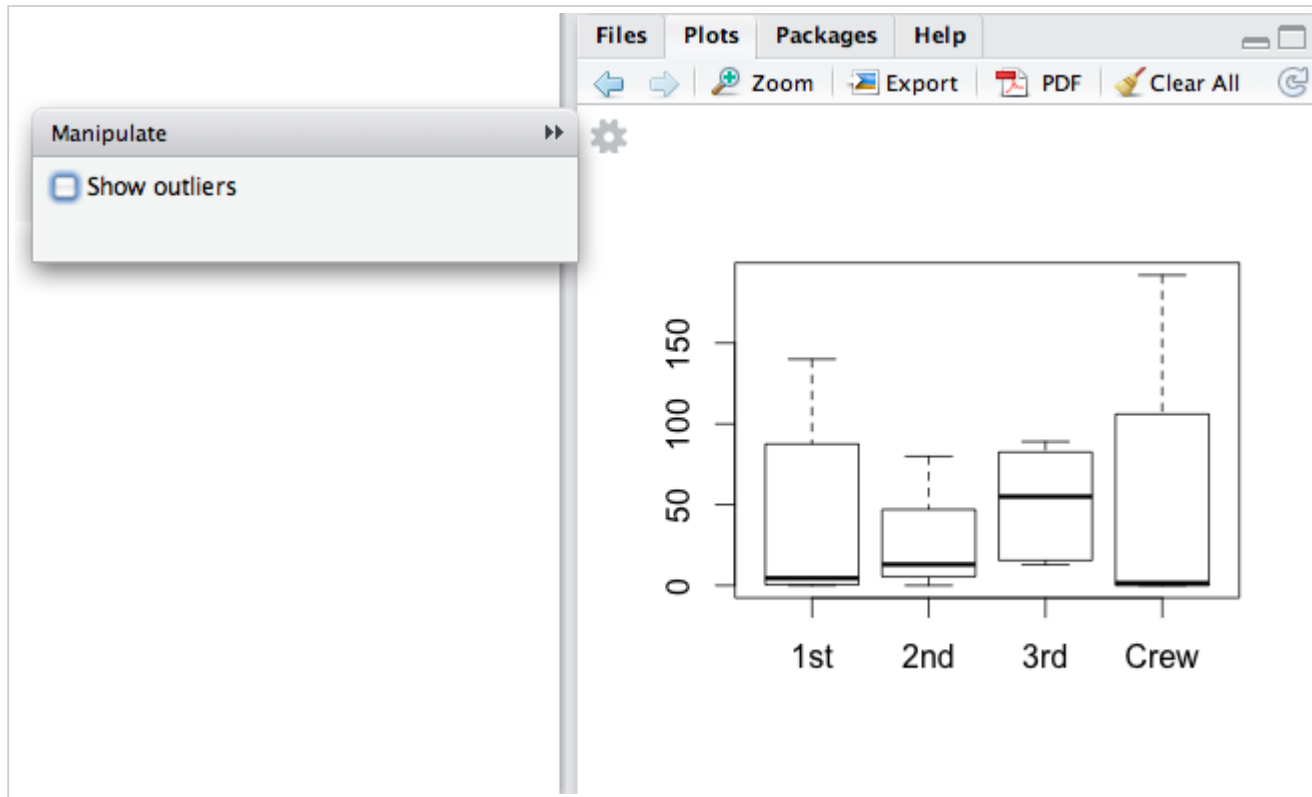
Picker controls support arbitrary value types, and can also include custom user-readable labels for each choice.

Checkbox Control

The checkbox control enables manipulation of logical plot variables. For example:

```
manipulate(  
  boxplot(Freq ~ Class, data = Titanic, outline = outline),  
  outline = checkbox(FALSE, "Show outliers"))
```

Results in this plot and manipulator:



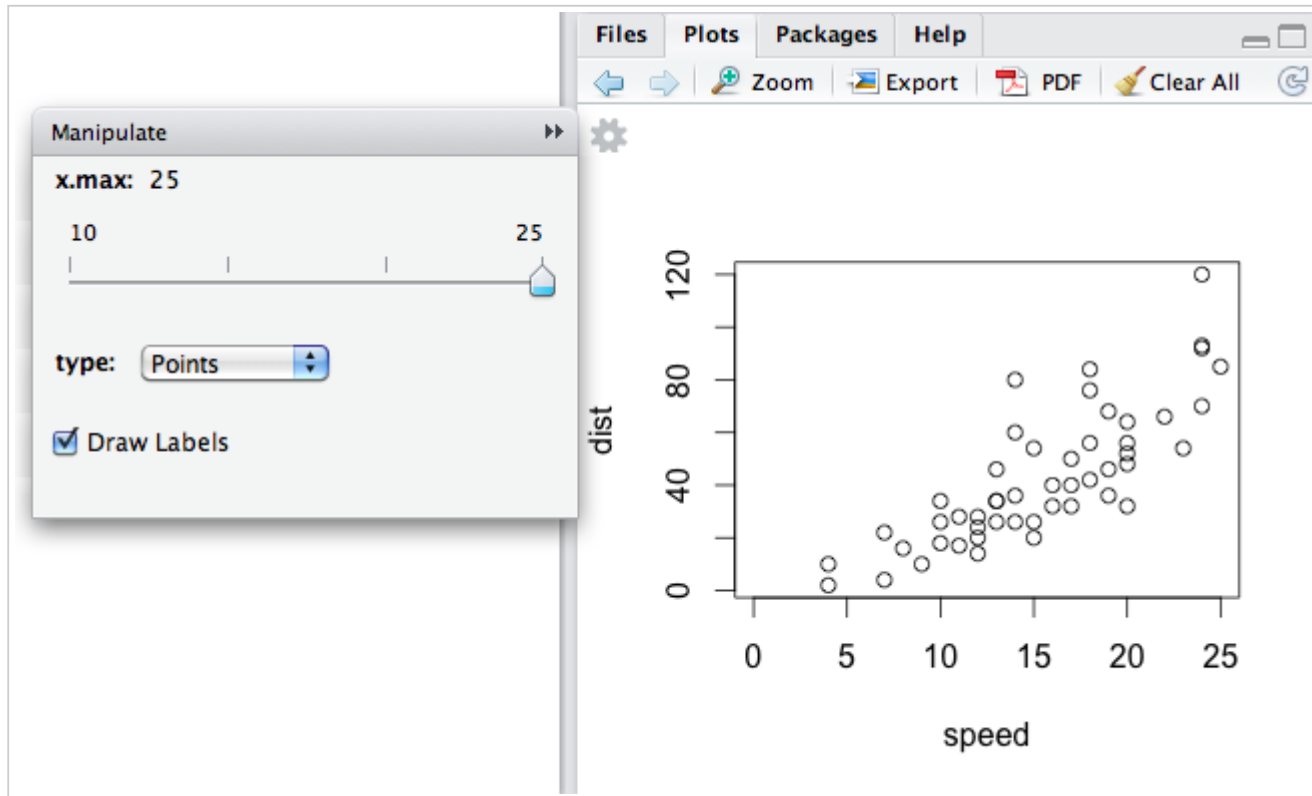
The `manipulate` package documentation contains additional details on all of the options available for the various control types.

Combining Controls

Multiple controls can be combined within a single manipulator. For example:

```
manipulate(  
  plot(cars, xlim = c(0, x.max), type = type, ann = label),  
  x.max = slider(10, 25, step=5, initial = 25),  
  type = picker("Points" = "p", "Line" = "l", "Step" = "s"),  
  label = checkbox(TRUE, "Draw Labels"))
```

Results in this plot and manipulator:



© 2011 RStudio, Inc.

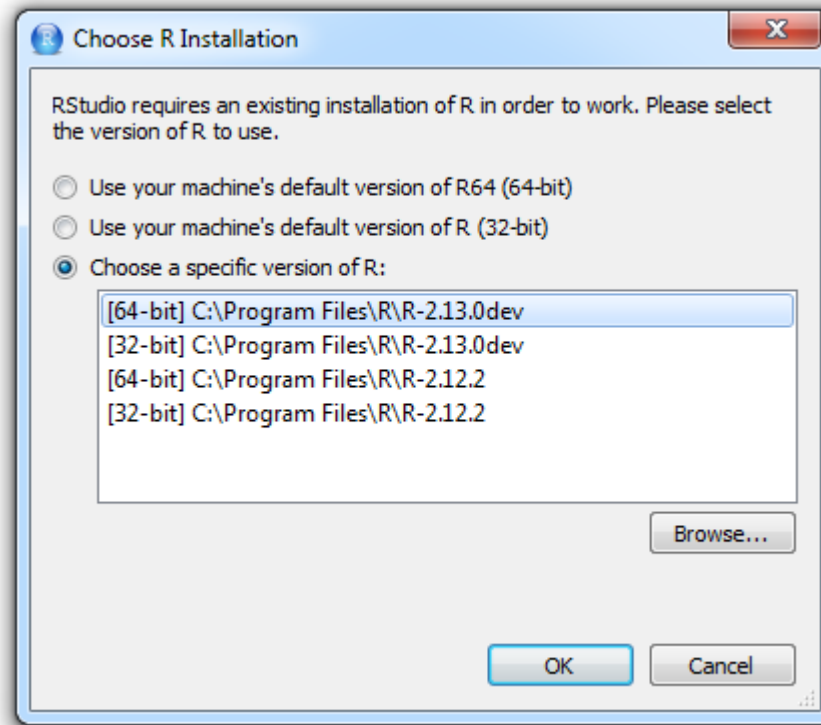
Using Different Versions of R

RStudio requires R version 2.11.1 or higher. Since R versions can be installed side-by-side on a system, RStudio needs to select which version of R to run against. The way this occurs varies between platforms—this article covers how version selection is handled on each platform.

Windows

On Windows, RStudio uses the system's current version of R by default. When R is installed on Windows it writes the version being installed to the Registry as the "current" version of R (the specific registry keys written are described [here](#)). This is the version of R which RStudio runs against by default.

You can override which version of R is used via General panel of the RStudio Options dialog. This dialog allows you to specify that RStudio should always bind to the default 32 or 64-bit version of R, or to specify a different version altogether:



Note that by holding down the Control key during the launch of RStudio you can cause the R version selection dialog to display at startup.

Mac OS X

R from CRAN

On Mac OS X if the only version of R you have installed is the standard R distribution from CRAN then RStudio will by default run against the current version of R.Framework. You can list all of the versions of R.Framework on your system and determine which one is considered the current one by executing the following command:

```
ls -l /Library/Frameworks/R.framework/Versions/
```

To change the current version of R.Framework you can either:

- Run the installer from CRAN for the R version you want to be current
- Use the RSwitch utility available at: <http://r.research.att.com/>
- Update the R.framework/Versions/Current directory alias directly using `ln -s`

R from source (including MacPorts and Homebrew)

When R is installed from CRAN on OS X the R executable is installed at `/usr/bin/R`. However, if R is installed directly from source or via a package manager like MacPorts or Homebrew, then the R executable is installed to either `/usr/local/bin/R` (Homebrew) or `/opt/local/bin/R` (MacPorts). In order to support these variations, RStudio scans for the R executable in the following sequence:

1. `/opt/local/bin/R`
2. `/usr/bin/R`
3. `/usr/local/bin/R`

This order is based on the conventional ordering of the OS X PATH environment variable, and therefore should normally yield the same version that is run when R is executed from a terminal.

If you want to override the version of R selected by RStudio's default behavior then you can set the `RSTUDIO_WHICH_R` environment variable to the R executable that you want to run against. For example, to force RStudio to use the R executable located at `/usr/local/bin/`:

```
export RSTUDIO_WHICH_R=/usr/local/bin/R
```

Note that in order for RStudio to see this environment variable it needs to be added to the OS X `environment.plist` file. Instructions for editing this file are [available here](#).

Linux

On Linux, RStudio uses the version of R pointed to by the output of the following command:

```
which R
```

The `which` command performs a search for the R executable using the system PATH. RStudio will therefore by default bind to the same version that is run when R is executed from a terminal.

For versions of R installed by system package managers (e.g. `r-base` on Ubuntu) this will be `/usr/bin/R`. For versions of R installed from source this will typically (but not always) be `/usr/local/bin/R`.

If you want to override which version of R is used then you can set the `RSTUDIO_WHICH_R` environment variable to the R executable that you want to run against. For example:

```
export RSTUDIO_WHICH_R=/usr/local/bin/R
```

Not that in order for RStudio to see this environment variable when launched from the Ubuntu desktop Applications menu (as opposed to from a terminal) it must be defined in the `~/.profile` file.

Web

If you are running RStudio within a web browser then the version of R is determined by whatever version of R is running alongside RStudio Server. The version currently in use on the server can be printed using the following command:

```
> R.version.string
```

Character Encoding

Starting with version 0.93, RStudio supports non-ASCII characters for input and output.

Console

Unicode characters can be used for both input and output in the console.

Source Editor

The RStudio source editor natively supports Unicode characters. It will allow you to type or paste characters from any language, even ones that are not part of the document's character set. RStudio will allow you to save such documents, but will print a warning to the R console that not all characters could be encoded. If you close the document without re-saving in a more suitable encoding, those characters will be lost.

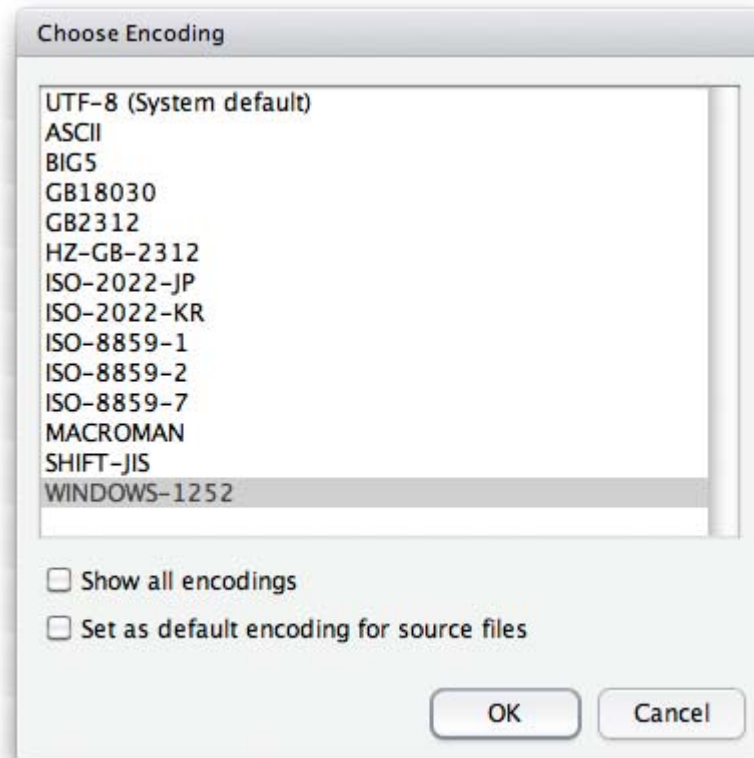
If in doubt about which encoding to use, use UTF-8, as it can encode any Unicode character.

Reading and Writing Files

The RStudio source editor can read and write files using any character encoding that is available on your system:

- You can choose the encoding for reading with **File : Reopen with Encoding**, which will re-read the current file from disk with the new encoding.
- You can also save an open file using a different encoding with **File : Save with Encoding**.

The Reopen and Save with Encoding commands both display the following dialog:



Setting the Default Encoding

If you frequently use the character set, check "Set as default encoding for source files". You can view or change this default in the **Tools : Options** (for Windows & Linux) or **Preferences** (for Mac) dialog, in the **Editing** section.

If you don't set a default encoding, files will be opened using UTF-8 (on Mac desktop, Linux desktop, and server) or the system's default encoding (on Windows). When saving a previously unsaved file, RStudio will ask you to choose an encoding if non-ASCII characters are present.

Known Issues

- If you call `Sys.setlocale` with "LC_CTYPE" or "LC_ALL" to change the system locale while RStudio is running, you may run into some minor issues as RStudio assumes the system encoding doesn't change. If you are on Windows, we recommend you only call `Sys.setlocale` in `.Rprofile`. If you are on Mac or Linux and want to change the system locale, please visit the [support forum](#) and let us know your scenario.

- On Windows, R's `source` function does not work with files that include characters that aren't part of the current system encoding. You may have trouble with RStudio's **Run All** and **Source on Save** commands, as they rely on `source`.

© 2011 RStudio, Inc.

Optimizing your Browser for RStudio

NOTE: This article is only applicable if you are using the RStudio IDE within a web browser (as opposed to using RStudio as a standalone desktop application).

Run a Recent Browser Version

RStudio makes use of a number of advanced web browser features and as a result benefits from running within more up-to-date browser versions. The following are the recommend minimum versions for various browsers:

- [Firefox 3.5](#)
- [Safari 4.0](#)
- [Google Chrome 5.0](#)

Note that RStudio can also be run from within Internet Explorer using the [Google Chrome Frame](#) browser plugin.

Disable Pop-Up Blockers

There are a number of instances where RStudio needs to show a new external popup window (e.g to display a PDF file). We therefore recommend that you disable pop-up blocking for the RStudio domain. Most browsers will prompt you regarding whether you want to enable popups for RStudio the first time one is blocked. Some browsers (such as Safari) may require you to globally enable and disable popups.

Safari and Chrome: Disable Browser Spell Checking

If you are using Safari or Chrome, you may find it desirable to disable "Check Spelling While Typing" (available from the Edit menu) since many of the words you enter in the Source and Console will not be in the built-in dictionary and thus will show up with a red underline when entered.

Firefox for Mac: Install Inline PDF Extension

Firefox for the Mac does not display PDFs inline by default (rather they are downloaded like any other file). RStudio uses PDFs for both printing plots as well as for Sweave/LaTeX documents and having them display inline is much preferred. To enable this you should install the following Firefox extension: <http://code.google.com/p/firefox-mac-pdf/>

© 2011 RStudio, Inc.

Uploading and Downloading Files

NOTE: This article is only applicable if you are using the RStudio IDE within a web browser (as opposed to using RStudio as a standalone desktop application).

Uploading Files

To upload datasets, scripts, or other files to RStudio Server you should take the following steps:

1. Switch to the **Files** pane
2. Navigate to the directory you wish to upload files into
3. Click the **Upload** toolbar button
4. Choose the file you wish to upload and press OK

Note that if you wish to upload several files or even an entire folder, you should first compress your files or folder into a zip file and then upload the zip file (when RStudio receives an uploaded zip file it automatically uncompresses it).

Downloading Files

To download files from RStudio Server you should take the following steps:

1. Switch to directory you want to download files from within the **Files** pane
2. Select the file(s) and/or folder(s) you want to download
3. Click **More** -> **Export** on the toolbar
4. You'll then be prompted with a default file name for the download. Either accept the default or specify a custom name then press OK.

Note that if you select multiple files or folders for download then RStudio compresses all of the files into a single zip archive for downloading.