ESS – Emacs Speaks Statistics

ESS version 18.10

**The ESS Developers (A.J. Rossini, R.M. Heiberger, K. Hornik,
M. Maechler, R.A. Sparapani, S.J. Eglen,
S.P. Luque, H. Redestig, V. Spinu, L. Henry, and J.A. Branham)
Current Documentation by The ESS Developers**

Table of Contents

1 Introduction to ESS

ESS provides a generic interface, through Emacs, to statistical packages. It currently supports R (and the rest of the S family), SAS, BUGS/JAGS, Stata, and Julia with the level of support roughly in that order.

Throughout this manual, *Emacs* refers to *GNU Emacs* by the Free Software Foundation. Although previous versions of ESS supported other Emacsen, current versions only support GNU Emacs.

There are two main ways of interacting with ESS: through "regular" modes or "inferior" modes. Regular modes act like normal Emacs major modes. ESS major mods are displayed in the mode-line in the format `ESS[dialect]`, where `dialect` can take values such as `R`, `SAS`, or `S`.

ESS also provides easy access to an "inferior process," which is an Emacs buffer associated with a running process. This can be an R session, for example. These inferior processes are referred to as inferior ESS (`iESS`), and are shown in the modeline by `iESS [dialect]`.

Currently, the documentation contains many references to *'R'* where actually any supported (statistics) language is meant, i.e., 'R' could also mean 'S' or 'SAS.'

For exclusively interactive users of R, ESS provides a number of features to make life easier. There is an easy to use command history mechanism, including a quick prefix-search history. To reduce typing, command-line completion is provided for all R objects and "hot keys" are provided for common R function calls. Help files are easily accessible, and a paging mechanism is provided to view them. Finally, an incidental (but very useful) side-effect of ESS is that a transcript of your session is kept for later saving or editing.

No special knowledge of Emacs is necessary when using R interactively under ESS.

For those that use R in the typical edit–test–revise cycle when writing R functions, ESS provides for editing of R functions in Emacs buffers. Unlike the typical use of R where the editor is restarted every time an object is edited, ESS uses the current Emacs session for editing. In practical terms, this means that you can edit more than one function at once, and that the ESS process is still available for use while editing. Error checking is performed on functions loaded back into R, and a mechanism to jump directly to the error is provided. ESS also provides for maintaining text versions of your R functions in specified source directories.

1.1 Why should I use ESS?

Statistical packages are powerful software systems for manipulating and analyzing data, but their user interfaces often leave something something to be desired: they offer weak editor functionality and they differ among themselves so markedly that you have to re-learn how to do those things for each package. ESS is a package which is designed to make editing and interacting with statistical packages more uniform, user-friendly and give you the power of Emacs as well.

Additionally, both Emacs and ESS (and R) are free software designed to give users full control over their computer. For more on what this means, visit `https://www.gnu.org/philosophy/free-sw.html`.

1.1.1 Features Overview

- Languages Supported:
  - S family (R, S, and S+ AKA S-PLUS)
  - SAS
  - BUGS/JAGS
  - Stata
  - Julia
- Editing source code (S family, SAS, BUGS/JAGS, Stata, Julia)
  - Syntactic indentation and highlighting of source code
  - Partial evaluation of code
  - Loading and error-checking of code
  - Source code revision maintenance
  - Batch execution (SAS, BUGS/JAGS)
  - Use of imenu to provide links to appropriate functions
- Interacting with the process (S family, SAS, Stata, Julia)
  - Command-line editing
  - Searchable Command history
  - Command-line completion of S family object names and file names
  - Quick access to object lists and search lists
  - Transcript recording
  - Interface to the help system
- Transcript manipulation (S family, Stata)
  - Recording and saving transcript files
  - Manipulating and editing saved transcripts
  - Re-evaluating commands from transcript files
- Interaction with Help Pages and other Documentation (R)
  - Fast Navigation
  - Sending Examples to running ESS process.
  - Fast Transfer to Further Help Pages
- Help File Editing (R)
  - Syntactic indentation and highlighting of source code.
  - Sending Examples to running ESS process.
  - Previewing

For source code buffers, ESS offers several features:

- **Support for multiple indentation styles** R code See Section 7.4 [Indenting], page 28.
- **Facilities for loading and error-checking source files**, including a keystroke to jump straight to the position of an error in a source file. See Section 7.3 [Error Checking], page 28.

- **Source code revision maintenance**, which allows you to keep historic versions of R source files. See Section 7.6 [Source Files], page 30.
- **Facilities for evaluating R code** such as portions of source files, or line-by-line evaluation of files (useful for debugging). See Chapter 5 [Evaluating code], page 24.

ESS also provides features that make it easier to interact with inferior ESS (iESS) process (a connection between your buffer and the statistical package which is waiting for you to input commands). These include:

- **Command-line editing** for fixing mistakes in commands before they are entered. See Section 4.1 [Command-line editing], page 15.
- **Searchable command history** for recalling previously-submitted commands. See Section 4.3 [Command History], page 18.
- **Command-line completion** of both object and file names for quick entry. See Chapter 9 [Completion], page 36.
- **Hot-keys** for quick entry of commonly-used commands in 'R' such as `objects()`, and `search()`. See Section 4.5 [Hot keys], page 20.
- **Transcript recording** for a complete record of all the actions in an R session. See Section 4.2 [Transcript], page 15.
- **Interface to the help system**, with a specialized mode for viewing R help files. See Chapter 8 [Help], page 34.
- **Object editing**. ESS allows you to edit more than one function simultaneously in dedicated Emacs buffers. The ESS process may continue to be used while functions are being edited. See Section 7.1 [Edit buffer], page 27.

Finally, ESS provides features for re-submitting commands from saved transcript files, including:

- **Evaluation of previously entered commands**, stripping away unnecessary prompts. See Section 4.2.3 [Transcript resubmit], page 16.

1.2 New features in ESS

Changes and New Features in 18.10:

- This is the last release to support Emacs older than 25.1. Going forward, only GNU Emacs 25.1 and newer will be supported. Soon after this release, support for older Emacs versions will be dropped from the git master branch. Note that MELPA uses the git master branch to produce ESS snapshots, so if you are using Emacs < 25.1 from MELPA and are unable to upgrade, you should switch to MELPA-stable.
- ESS now displays the language dialect in the mode-line. So, for example, R buffers will now show ESS[R] rather than ESS[S].
- The ESS manual has been updated and revised.
- The ESS initialization process has been further streamlined. If you update the autoloads (which installation from `package-install` does), you should not need to `(require 'ess-site)` at all, as autoloads should automatically load ESS when it is needed (e.g. the first time an R buffer is opened). In order to defer loading your ESS config, you may want to do something like `(with-require-after-load "ess" <ess-config-here>)` in your Emacs init file. Users of the popular `use-package` Emacs

package can now do `(use-package ess :defer t)` to take advantage of this behavior. See Section "Activating and Loading ESS" in `ess` for more information on this feature.

- ESS now respects Emacs conventions for keybindings. This means that The `C-c [letter]` bindings have been removed. This affects `C-c h`, which was bound to `ess-eval-line-and-step-invisibly` in `sas-mode-local-map`; `C-c f`, which was bound to `ess-insert-function-outline` in `ess-add-MM-keys`; and `C-c h`, which was bound to `ess-handy-commands` in `Rd-mode-map`, `ess-noweb-minor-mode-map`, and `ess-help-mode-map`
- ESS[R]: `ess-r-package-use-dir` now works with any mode. This sets the working directory to the root of the current package including for example C or C++ files within `/src`).
- ESS[R]: Long `+ +` prompts in the inferior no longer offset output.
- ESS[R]: New option `strip` for `inferior-ess-replace-long+`. This strips the entire `+ +` sequence.
- ESS modes now inherit from `prog-mode`. In the next release, ESS modes will use `define-derived-mode` so that each mode will have (for example) its own hooks and keymaps.
- ESS[R]: Supports flymake in R buffers for Emacs 26 and newer. Users need to install the `lintr` package to use it. Customizable options include `ess-use-flymake`, `ess-r-flymake-linters`, and `ess-r-flymake-lintr-cache`.
- ESS[R]: Gained support for xref in Emacs 25+. See Section "Xref" in *The Gnu Emacs Reference Manual*
- ESS[R]: The startup screen is cleaner. It also displays the startup directory with an explicit `setwd()`.
- ESS[R]: Changing the working directory is now always reflected in the process buffer.
- ESS[R]: `Makevars` files open with `makefile-mode`.
- New variable `ess-write-to-dribble`. This allows users to disable the dribble (`*ESS*`) buffer if they wish.
- All of the `*-program-name` variables have been renamed to `*-program`. Users who previously customized e.g. `inferior-ess-R-program-name` will need to update their customization to `inferior-ess-R-program`. These variables are treated as risky variables.
- `ess-smart-S-assign` was renamed to `ess-insert-assign`. It provides similar functionality but for any keybinding, not just '_'. For instance if you bind it to ';', repeated invokations cycle through between assignment and inserting ';'.
- `C-c C-=` is now bound to `ess-cycle-assign` by default. See the documentation for details. New user customization option `ess-assign-list` controls which assignment operators are cycled.
- ESS[R] In remote sessions, the ESSR package is now fetched from GitHub.
- Commands that send the region to the inferior process now deal with rectangular regions. See the documentation of `ess-eval-region` for details. This only works on Emacs 25.1 and newer.

- ESS[R]: Improvements to interacting with iESS in non-R files. Interaction with inferior process in non-R files within packages (for instance C or C++ files) has been improved. This is a work in progress.
- ESS[R]: Changing the working directory is now always reflected in the process buffer.
- ESS[JAGS]: *.jog and *.jmd files no longer automatically open in JAGS mode.

Many improvements to fontification:

- Improved customization for faces. ESS now provides custom faces for (nearly) all faces used and places face customization options into their own group. Users can customize these options using *M-x customize-group RET ess-faces*.
- Many new keywords were added to `ess-R-keywords` and `ess-R-modifiers`. See the documentation for details.
- ESS[R]: `in` is now only fontified when inside a `for` construct. This avoids spurious fontification, especially in the output buffer where 'in' is a commond English word.
- ESS: Font-lock keywords are now generated lazily.That means you can now add or remove keywords from variables like `ess-R-keywords` in your Emacs configuration file after loading ESS (i.e. in the `:config` section for `use-package` users).
- ESS[R]: Fontification of roxygen `@param` keywords now supports comma-separated parameters.
- ESS[R]: Certain keywords are only fontified if followed by a parenthesis. Function-like keywords such as `if ()` or `stop()` are no longer fontified as keyword if not followed by an opening parenthesis. The same holds for search path modifiers like `library()` or `require()`.
- ESS[R]: Fixed fontification toggling. Especially certain syntactic elements such as `%op%` operators and backquoted function definitions.
- ESS[R]: `ess-font-lock-toggle-keyword` can be called interactively. This command asks with completion for a font-lock group to toggle. This functionality is equivalent to the font-lock menu.

Notable bug fixes:

- `prettify-symbols-mode` no longer breaks indentation. This is accomplished by having the pretty symbols occupy the same number of characters as their non-pretty cousins. You may customize the new variable `ess-r-prettify-symbols` to control this behavior.
- ESS: Inferior process buffers are now always displayed on startup. Additionally, they don't hang Emacs on failures.

Obsolete libraries, functions, and variables:

- The `ess-r-args.el` library has been obsoleted and will be removed in the next release. Use `eldoc-mode` instead, which is on by default.
- Functions and options dealing with the smart assign key are obsolete. The following functions have been made obsolete and will be removed in the next release of ESS: `ess-smart-S-assign`, `ess-toggle-S-assign`, `ess-toggle-S-assign-key`, `ess-disable-smart-S-assign`.

The variable `ess-smart-S-assign-key` is now deprecated and will be removed in the next release. If you would like to continue using '_' for insterting assign in future releases, please bind `ess-insert-assign` in `ess-mode-map` the normal way.

- ESS[S]: Variable `ess-s-versions-list` is obsolete and ignored. Use `ess-s-versions` instead. You may pass arguments by starting the inferior process with the universal argument.

Changes and New Features in 17.11:

- The ESS initialisation process has been streamlined. You can now load the R and Stata modes independently from the rest of ESS. Just put `(require 'ess-r-mode)` or `(require 'ess-stata-mode)` in your init file. This is for experienced Emacs users as this requires setting up autoloads for `.R` files manually. We will keep maintaining `ess-site` for easy loading of all ESS features.
- Reloading and quitting the process is now more robust. If no process is attached, ESS now switches automatically to one (prompting you for selection if there are several running). Reloading and quitting will now work during a debug session or when R is prompting for input (for instance after a crash). Finally, the window configuration is saved and restored after reloading to prevent the buffer of the new process from capturing the cursor.
- ESS[R]: New command `ess-r-package-use-dir`. It sets the working directory of the current process to the current package directory.
- ESS[R] Lookup for references in inferior buffers has been improved. New variable `ess-r-package-source-roots` contains package sub-directories which are searched recursively during the file lookup point. Directories in `ess-tracebug-search-path` are now also searched recursively.
- ESS[R] Namespaced evaluation is now automatically enabled only in the `R/` directory. This way ESS will not attempt to update function definitions from a package if you are working from e.g. a test file.

Changes and New Features in 16.10:

- ESS[R]: Syntax highlighting is now more consistent. Backquoted names are not fontified as strings (since they really are identifiers). Furthermore they are now correctly recognised when they are function definitions or function calls.
- ESS[R]: Backquoted names and `%op%` operators are recognised as sexp. This is useful for code navigation, e.g. with *`C-M-f`* and *`C-M-b`*.
- ESS[R]: Integration of outline mode with roxygen examples fields. You can use outline mode's code folding commands to fold the examples field. This is especially nice to use with well documented packages with long examples set. Set `ess-roxy-fold-examples` to non-nil to automatically fold the examples field when you open a buffer.
- ESS[R]: New experimental feature: syntax highlighting in roxygen examples fields. This is turned off by default. Set `ess-roxy-fontify-examples` to non-nil to try it out.
- ESS[R]: New package development command `ess-r-devtools-ask` bound to *`C-c C-w C-a`*. It asks with completion for any devtools command that takes `pkg` as argument.
- ESS[R]: New command *`C-c C-e C-r`* to reload the inferior process. Currently only implemented for R. The R method runs `inferior-ess-r-reload-hook` on reloading.

- ESS[R]: `ess-r-package-mode` is now activated in non-file buffers as well.

Bug fixes in 16.10:

- ESS[R]: Fix broken (un)flagging for debugging inside packages
- ESS[R]: Fixes (and improvements) in Package development
- ESS[R]: Completion no longer produces `...=` inside `list( )`.
- ESS[R]: Better debugging and tracing in packages.
- ESS[R]: Better detection of symbols at point.
- ESS[R]: No more spurious warnings on deletion of temporary files.
- ESS[julia]: help and completion work (better)
- ESS[julia]: available via `ess-remote`

Changes and New Features in 16.04:

- ESS[R]: `developer` functionality has been refactored. The new user interface consists of a single command `ess-r-set-evaluation-env` bound by default to *C-c C-t C-s*. Once an evaluation environment has been set with, all subsequent ESS evaluation will source the code into that environment. By default, for file within R packages the evaluation environment is set to the package environment. Set `ess-r-package-auto-set-evaluation-env` to `nil` to disable this.
- ESS[R]: New `ess-r-package-mode` This development mode provides features to make package development easier. Currently, most of the commands are based on the `devtools` packages and are accessible with *C-c C-w* prefix. See the documentation of `ess-r-package-mode` function for all available commands. With *C-u* prefix each command asks for extra arguments to the underlying devtools function. This mode is automatically enabled in all files within R packages and is indicated with `[pkg:NAME]` in the mode-line.
- ESS[R]: Help lookup has been improved. It is now possible to get help for namespaced objects such as pkg::foobar. Furthermore, ESS recognizes more reliably when you change `options('html_type')`.
- ESS[R]: New specialized breakpoints for debugging magrittr pipes
- ESS: ESS now implements a simple message passing interface to communicate between ESS and inferior process.

Bug fixes in 16.04:

- ESS[R]: Roxygen blocks with backtics are now correctly filled
- ESS[R]: Don't skip breakpoints in magrittr's `debug_pipe`
- ESS[R]: Error highlighting now understands 'testthat' type errors
- ESS[Julia]: Added getwd and setwd generic commands

1.3 Authors of and contributors to ESS

The ESS environment is built on the open-source projects of many contributors, dating back to 1989 where Doug Bates and Ed Kademan wrote S-mode to edit S and Splus files in GNU Emacs. Frank Ritter and Mike Meyer added features, creating version 2. Meyer and David Smith made further contributions, creating version 3. For version 4, David Smith provided significant enhancements to allow for powerful process interaction.

John Sall wrote GNU Emacs macros for SAS source code around 1990. Tom Cook added functions to submit jobs, review listing and log files, and produce basic views of a dataset, thus creating a SAS-mode which was distributed in 1994.

In 1994, A.J. Rossini extended S-mode to support XEmacs. Together with extensions written by Martin Maechler, this became version 4.7 and supported S, Splus, and R. In 1995, Rossini extended SAS-mode to work with XEmacs.

In 1997, Rossini merged S-mode and SAS-mode into a single Emacs package for statistical programming; the product of this marriage was called ESS version 5. Richard M. Heiberger designed the inferior mode for interactive SAS and SAS-mode was further integrated into ESS. Thomas Lumley's Stata mode, written around 1996, was also folded into ESS. More changes were made to support additional statistical languages, particularly XLispStat.

ESS initially worked only with Unix statistics packages that used standard-input and standard-output for both the command-line interface and batch processing. ESS could not communicate with statistical packages that did not use this protocol. This changed in 1998 when Brian Ripley demonstrated use of the Windows Dynamic Data Exchange (DDE) protocol with ESS. Heiberger then used DDE to provide interactive interfaces for Windows versions of Splus. In 1999, Rodney A. Sparapani and Heiberger implemented SAS batch for ESS relying on files, rather than standard-input/standard-output, for Unix, Windows and Mac. In 2001, Sparapani added BUGS batch file processing to ESS for Unix and Windows.

- The multiple process code, and the idea for `ess-eval-line-and-next-line` are by Rod Ball.
- Thanks to Doug Bates for many useful suggestions.
- Thanks to Martin Maechler for reporting and fixing bugs, providing many useful comments and suggestions, and for maintaining the ESS mailing lists.
- Thanks to Frank Ritter for updates, particularly the menu code, and invaluable comments on the manual.
- Thanks to Ken'ichi Shibayama for his excellent indenting code, and many comments and suggestions.
- Thanks to Aki Vehtari for adding interactive BUGS support.
- Thanks to Brendan Halpin for bug-fixes and updates to Stata-mode.
- Last, but definitely not least, thanks to the many ESS users and contributors to the ESS mailing lists.

ESS is being developed and currently maintained by

- A.J. Rossini (`mailto:blindglobe@gmail.com`)
- Richard M. Heiberger (`mailto:rmh@temple.edu`)
- Kurt Hornik (`mailto:Kurt.Hornik@R-project.org`)
- Martin Maechler (`mailto:maechler@stat.math.ethz.ch`)
- Rodney A. Sparapani (`mailto:rsparapa@mcw.edu`)
- Stephen Eglen (`mailto:stephen@gnu.org`)
- Sebastian P. Luque (`mailto:spluque@gmail.com`)
- Henning Redestig (`mailto:henning.red@googlemail.com`)
- Vitalie Spinu (`mailto:spinuvit@gmail.com`)

- Lionel Henry (mailto:lionel.hry@gmail.com)
- J. Alexander Branham (mailto:alex.branham@gmail.com)

1.4 How to read this manual

If you need to install ESS, read Chapter 2 [Installation], page 10, for details on what needs to be done before proceeding to the next chapter. This section describes some of the basics of using Emacs. If you are already familiar with basic Emacs functionality, skip this section. You may also want to use the Emacs tutorial, accessible via *C-h t*.

In this manual we use the standard notation used by Emacs for describing the keystrokes used to invoke certain commands. *C-<chr>* means hold the CONTROL key while typing the character <chr>. *M-<chr>* means hold the META key (usually ALT) down while typing <chr>. If there is no META, EDIT or ALT key, instead press and release the ESC key and then type <chr>.

All ESS commands can be invoked by typing *M-x command*. Most of the useful commands are bound to keystrokes for ease of use. Also, the most popular commands are also available through the Emacs menubar, and a small subset are provided on the toolbar. Where possible, keybindings are similar to other modes in Emacs to strive for a consistent user interface within Emacs, regardless of the details of which programming language is being edited, or process being run.

Some commands, such as *M-x R* can accept an optional 'prefix' argument. To specify the prefix argument, you would type *C-u* before giving the command. For example, if you type *C-u M-x R*, you will be asked for command line options that you wish to invoke the R process with.

Emacs is a 'self-documenting' text editor. This applies to ESS in two ways. First, some documentation about each ESS command can be obtained by typing *C-h f*. For example, if you type *C-h f ess-eval-region*, documentation for that command will appear in a separate *Help* buffer. Second, *C-h m* pops up a complete list of keybindings available in each ESS mode and brief description of that mode.

Emacs is a versatile editor written in both C and a dialect of lisp known as elisp. ESS is written in elisp and benefits from the flexible nature of lisp. In particular, many aspects of ESS behaviour can be changed by suitable customization of lisp variables. This manual mentions some of the most frequent variables. A full list of them however is available by using the Custom facility within Emacs. Type *M-x customize-group RET ess RET* to get started. Appendix A [Customization], page 68, provides details of common user variables you can change to customize ESS to your taste, but it is recommended that you defer this section until you are more familiar with ESS.

2 Installing ESS on your system

ESS supports GNU Emacs versions 24.3 and newer.

ESS is most likely to work with current/recent versions of the following statistical packages: R/S-PLUS, SAS, Stata, OpenBUGS and JAGS.

To build the PDF documentation, you will need a version of TeX Live or texinfo that includes texi2dvi.

There are two main methods used for installing ESS. You may install from a third-party repository or from source code. Once you install it, you must also activate or load ESS in each Emacs session, though installation from a third-party repository likely takes care of that for you. See Section 2.3 [Activating and Loading ESS], page 11, for more details.

2.1 Installing from a third-party repository

ESS is packaged by many third party repositories. Many GNU/Linux distributions package it, usually with the name "emacs-ess" or similar.

ESS is also available through Milkypostmans Emacs Lisp Package Archive (MELPA), a popular repository for Emacs packages. Instructions on how to do so are found on MELPA's website (`https://melpa.org/`). MELPA also hosts MELPA-stable with stable ESS builds. You may choose between MELPA with the latest and greatest features (and bugs) or MELPA-stable, which may lag a bit behind but should be more stable.

After installing, users should make sure ESS is activated or loaded in each Emacs session. See Section 2.3 [Activating and Loading ESS], page 11. Depending on install method, this may be taken care of automatically.

2.2 Installing from source

Stable versions of ESS are available at the ESS web page (`https://ess.r-project.org`) as a .tgz file or .zip file. ESS releases are GPG-signed, you should check the signature by downloading the accompanying `.sig` file and doing:

```
gpg --verify ess-18.10.tgz.sig
```

Alternatively, you may download the git repository. ESS is currently hosted on Github: `https: / / github . com / emacs-ess / ESS`. `git clone https://github.com/emacs-ess/ESS.git` will download it to a new directory `ESS` in the current working directory.

We will refer to the location of the ESS source files as `/path/to/ESS/` hereafter.

After installing, users should make sure they activate or load ESS in each Emacs session, see Section 2.3 [Activating and Loading ESS], page 11,

Optionally, compile elisp files, build the documentation, and the autoloads:

```
cd /path/to/ESS/
make
```

Without this step the documentation, reference card, and autoloads will not be available. Uncompiled ESS will also run slower.

Optionally, you may make ESS available to all users of a machine by installing it site-wide. To do so, run `make install`. You might need administrative privileges:

```
make install
```

The files are installed into `/usr/share/emacs` directory. For this step to run correctly on macOS, you will need to adjust the `PREFIX` path in `Makeconf`. The necessary code and instructions are commented in that file.

2.3 Activating and Loading ESS

After installing ESS, you must activate or load it each Emacs session. ESS can be autoloaded, and if you used a third-party repository (such as your Linux distribution or MELPA) to install, you can likely skip this section and proceed directly to Section 2.4 [Check Installation], page 11,

Otherwise, you may need to add the path to ESS to `load-path` with:

```
(add-to-list 'load-path "/path/to/ESS/lisp")
```

You then need to decide whether to take advantage of deferred loading (which will result in a faster Emacs startup time) or require ESS when Emacs is loaded. To autoload ESS when needed (note that if installed from source, you must have run `make`):

```
(load "ess-autoloads")
```

To require ESS on startup, you can either put

```
(require 'ess-site)
```

or

```
(require 'ess-r-mode)
```

In your configuration file, depending on whether you want all ESS features or only R related features.

2.4 Check Installation

Restart Emacs and check that ESS was loaded from a correct location with `M-x ess-version`.

3 Interacting with statistical programs

As well as using ESS to edit your source files for statistical programs, you can use ESS to run these statistical programs. In this chapter, we mostly will refer by example to running R from within Emacs. The Emacs convention is to name such processes running under its control as 'inferior processes'. Some users find this terminology confusing; you may prefer to think of these as 'interactive processes.' Either way, we use the term 'iESS' to refer to the Emacs mode used to interact with statistical programs.

3.1 Starting an ESS process

To start an inferior R session on GNU/Linux, macOS, or Windows using the Cygwin bash shell, simply type *M-x R RET*. To start an R session on Windows when you use the MS-DOS/powershell shell, simply type *M-x S+6-msdos RET*. R will then (by default) ask the question

```
R starting data directory?
```

Enter the name of the directory you wish to have as the working directory (that is, the directory you wish to have `getwd()` return if using R).

You will then be popped into a buffer named '`*R*`' which will be used for interacting with the ESS process, and you can start entering commands.

3.2 Running more than one ESS process

ESS allows you to run more than one iESS process simultaneously in the same session. Each process has a name and a number; the initial process (process 1) is simply named '`R`'. If you call *M-x R* again without killing the first R process, ESS will start a second R process with the name '`R:2`'. To have the first buffer named '`R:1`', customize the option `ess-plain-first-buffername`. With a prefix argument, *C-u M-x R* allows for the specification of command line options.

`ess-plain-first-buffername` [User Option]

If non-`nil`, name the first iESS process [R]. Otherwise, name it [R:1].

You can switch to any active ESS process with the command '`M-x ess-request-a-process`'. Just enter the name of the process you require; completion is provided over the names of all running processes. This is a good command to consider binding to a global key.

3.3 ESS processes on Remote Computers

3.3.1 ESS and TRAMP

ESS works with processes on remote computers as easily as with processes on the local machine. The recommended way to access a statistical program on remote computer is to start it with See *TRAMP User Manual*.

Start an ssh session using TRAMP with '`C-x C-f /ssh:user@host: RET`'. Tramp should open a dired buffer in your remote home directory. Now call your favorite ESS process (`R`, `Julia`, `stata` etc) as you would usually do on local machine: `M-x R`.

Alternatively you can start your process normally (`M-x R`). When asked for starting directory, simply type '`/ssh:user@host: RET`'. The R process will be started on the remote machine.

To simplify the process even further create a "config" file in your `.ssh/` folder and add an account. For example if you use amazon EC2, it might look like following:

```
Host amazon
   Hostname ec2-54-215-203-181.us-west-1.compute.amazonaws.com
   User ubuntu
   IdentityFile ~/.ssh/my_amazon_key.pem
   ForwardX11 yes
```

With this configuration `/ssh:amazon:` is enough to start a connection. The ForwardX11 is needed if you want to see the R graphic device showing on the current machine

3.3.2 ESS-remote

TRAMP is the recommended way of starting a remote session. The other way to start a remote ESS connection is through `ess-remote`.

1. Start a new shell, telnet or ssh buffer and connect to the remote computer (e.g. use, '`M-x shell`', '`M-x telnet`' or '`M-x ssh`'; ssh.el is available at `https://www.splode.com/~friedman/software/emacs-lisp/src/ssh.el`).
2. Start the ESS process on the remote machine, for example with one of the commands '`R`', '`Splus`', or '`sas -stdio`'.
3. Start '`M-x ess-remote`'. You will be prompted for a program name with completion. Choose one. Your process is now known to ESS. All the usual ESS commands ('`C-c C-n`' and its relatives) now work with the R language processes. For SAS you need to use a different command '`C-c i`' (that is a regular '`i`', not a '`C-i`') to send lines from your `myfile.sas` to the remote SAS process. '`C-c i`' sends lines over invisibly. With ess-remote you get teletype behavior—the data input, the log, and the listing all appear in the same buffer. To make this work, you need to end every PROC and DATA step with a "RUN;" statement. The "RUN;" statement is what tells SAS that it should process the preceding input statements.
4. Graphics (interactive) on the remote machine. If you run X11 (See Section 11.6.2 [X11], page 47, X Windows) on both the local and remote machines then you should be able to display the graphs locally by setting the '`DISPLAY`' environment variable appropriately. Windows users can download '`xfree86`' from cygwin.
5. Graphics (static) on the remote machine. If you don't run the X window system on the local machine, then you can write graphics to a file on the remote machine, and display the file in a graphics viewer on the local machine. Most statistical software can write one or more of postscript, GIF, or JPEG files. Depending on the versions of Emacs and the operating system that you are running, Emacs itself may display '`.gif`' and '`.jpg`' files. Otherwise, a graphics file viewer will be needed. Ghostscript/ghostview may be downloaded to display '`.ps`' and '`.eps`' files. Viewers for GIF and JPEG are usually included with operating systems. See Section 13.5 [ESS(SAS)–Function keys for batch processing], page 58, for more information on using the F12 key for displaying graphics files with SAS.

Should you or a colleague inadvertently start a statistical process in an ordinary '`*shell*`' buffer, the '`ess-remote`' command can be used to convert it to an ESS buffer and allow you to use the ESS commands with it.

3.4 Changing the startup actions

If you do not wish ESS to prompt for a starting directory when starting a new process, set the variable `ess-ask-for-ess-directory` to `nil`. In this case, the starting directory will be set using one of the following methods:

1. If the variable `ess-directory-function` stores the name of a function, the value returned by this function is used. The default for this variable is nil.
2. Otherwise, if the variable `ess-directory` stores the name of a directory (ending in a slash), this value is used. The default for this variable is nil.
3. Otherwise, the working directory of the current buffer is used.

If `ess-ask-for-ess-directory` has a non-`nil` value (as it does by default) then the value determined by the above rules provides the default when prompting for the starting directory. Incidentally, `ess-directory` is an ideal variable to set in `ess-pre-run-hook`.

If you like to keep a record of your R actions, set the variable `ess-ask-about-transfile` to `t`, and you will be asked for a filename for the transcript before the ESS process starts.

`ess-ask-about-transfile` [User Option]
: If non-`nil`, ask for a file name in which to save the session transcript.

Enter the name of a file in which to save the transcript at the prompt. If the file doesn't exist it will be created (for R, you likely want it to end in '`.Rout`'). If the file already exists the transcript will be appended to the file. (Note: if you don't set this variable but you still want to save the transcript, you can still do it later — see Section 4.2.4 [Saving transcripts], page 17.)

Once these questions are answered (if they are asked at all) the inferior process itself is started. If you need to pass any arguments to this program, they may be specified in the variable `inferior-`*`S_program_name`*`-args`. For example, if `inferior-ess-program` is `"R"` then the variable to set is `inferior-R-args`. It is not normally necessary to pass arguments to the iESS program; in particular do not pass the '`-e`' option to `Splus`, since ESS provides its own command history mechanism.

By default, the new process will be displayed in the same window in the current frame. If you wish your iESS process to appear in a separate frame, customize the variable `inferior-ess-own-frame`. Alternatively, change `inferior-ess-same-window` if you wish the process to appear within another window of the current frame.

4 Interacting with the ESS process

The primary function of the ESS package is to provide an easy-to-use front end to the R interpreter. This is achieved by running the R process from within an Emacs buffer, called hereafter *inferior* buffer, which has an active `inferior-ess-mode`. The features of inferior R mode are similar to those provided by the standard Emacs shell mode (see Section "Shell Mode" in *The Gnu Emacs Reference Manual*). Command-line completion of R objects and a number of 'hot keys' for commonly-used R commands are also provided for ease of typing.

4.1 Entering commands and fixing mistakes

Sending a command to the ESS process is as simple as typing it in and pressing the `RETURN` key:

`inferior-ess-send-input` [Command]
`RET` Send the command on the current line to the ESS process.

If you make a typing error before pressing *`RET`* all the usual Emacs editing commands are available to correct it (see Section "Basic editing commands" in *The GNU Emacs Reference Manual*). Once the command has been corrected you can press `RETURN` (even if the cursor is not at the end of the line) to send the corrected command to the ESS process.

Emacs provides some other commands which are useful for fixing mistakes:

`C-c C-w` `backward-kill-word` Deletes the previous word (such as an object name) on the command line.

`C-c C-u` `comint-kill-input` Deletes everything from the prompt to point. Use this to abandon a command you have not yet sent to the ESS process.

`C-a` `comint-bol` Move to the beginning of the line, and then skip forwards past the prompt, if any.

See Section "Shell Mode" in *The Gnu Emacs Reference Manual*, for other commands relevant to entering input.

4.2 Manipulating the transcript

Most of the time, the cursor spends its time at the bottom of the ESS process buffer, entering commands. However all the input and output from the current (and previous) ESS sessions is stored in the process buffer (we call this the transcript) and often we want to move back up through the buffer, to look at the output from previous commands for example.

Within the process buffer, a paragraph is defined as the prompt, the command after the prompt, and the output from the command. Thus *`M-{`* and *`M-}`* move you backwards and forwards, respectively, through commands in the transcript. A particularly useful command is *`M-h`* (`mark-paragraph`) which will allow you to mark a command and its entire output (for deletion, perhaps). For more information about paragraph commands, see Section "Paragraphs" in *The GNU Emacs Reference Manual*.

If an ESS process finishes and you restart it in the same process buffer, the output from the new ESS process appears after the output from the first ESS process separated by a

form-feed (‘^L’) character. Thus pages in the ESS process buffer correspond to ESS sessions. Thus, for example, you may use *C-x [* and *C-x]* to move backward and forwards through ESS sessions in a single ESS process buffer. For more information about page commands, see Section “Pages” in *The GNU Emacs Reference Manual*.

4.2.1 Manipulating the output from the last command

Viewing the output of the command you have just entered is a common occurrence and ESS provides a number of facilities for doing this. Whenever a command produces output, it is possible that the window will scroll, leaving the next prompt near the middle of the window. The first part of the command output may have scrolled off the top of the window, even though the entire output would fit in the window if the prompt were near the bottom of the window. If this happens, you can use the following comint commands:

`comint-show-maximum-output` to move to the end of the buffer, and place cursor on bottom line of window to make more of the last output visible. To make this happen automatically for all inputs, set the variable `comint-scroll-to-bottom-on-input` to `t` or `'this`. If the first part of the output is still not visible, use *C-c C-r* (`comint-show-output`), which moves cursor to the previous command line and places it at the top of the window.

Finally, if you want to discard the last command output altogether, use *C-c C-o* (`comint-delete-output`), which deletes everything from the last command to the current prompt. Use this command judiciously to keep your transcript to a more manageable size.

4.2.2 Viewing older commands

If you want to view the output from more historic commands than the previous command, commands are also provided to move backwards and forwards through previously entered commands in the process buffer:

C-c C-p `comint-previous-prompt` Moves point to the preceding prompt in the process buffer.

C-c C-n `comint-next-prompt` Moves point to the next prompt in the process buffer.

Note that these two commands are analogous to *C-p* and *C-n* but apply to command lines rather than text lines. And just like *C-p* and *C-n*, passing a prefix argument to these commands means to move to the *ARG*’th next (or previous) command. (These commands are also discussed in Section “Shell History Copying” in *The GNU Emacs Reference Manual*.)

There are also two similar commands (not bound to any keys by default) which move to preceding or succeeding commands, but which first prompt for a regular expression (see Section “Syntax of Regular Expression” in *The GNU Emacs Reference Manual*), and then moves to the next (previous) command matching the pattern.

`comint-backward-matching-input regexp arg`
`comint-forward-matching-input regexp arg`
: Search backward (forward) through the transcript buffer for the *arg*’th previous (next) command matching *regexp*. *arg* is the prefix argument; *regexp* is prompted for in the minibuffer.

4.2.3 Re-submitting commands from the transcript

When moving through the transcript, you may wish to re-execute some of the commands you find there. ESS provides commands to do this; these commands may be used whenever

the cursor is within a command line in the transcript (if the cursor is within some command *output*, an error is signaled). Note all commands involve the `RETURN` key.

RET `inferior-ess-send-input` See Section 4.1 [Command-line editing], page 15.

C-c RET `comint-copy-old-input` Copy the command under the cursor to the current command line, but don't execute it. Leaves the cursor on the command line so that the copied command may be edited.

When the cursor is not after the current prompt, the `RETURN` key has a slightly different behavior than usual. Pressing *RET* on any line containing a command that you entered (i.e. a line beginning with a prompt) sends that command to the ESS process once again. If you wish to edit the command before executing it, use *C-c RET* instead; it copies the command to the current prompt but does not execute it, allowing you to edit it before (re)submitting it.

These commands work even if the current line is a continuation line (i.e. the prompt is '`+`' instead of '`>`') — in this case all the lines that form the multi-line command are concatenated together and the resulting command is sent to the ESS process (currently this is the only way to resubmit a multi-line command to the ESS process in one go). If the current line does not begin with a prompt, an error is signalled. This feature, coupled with the command-based motion commands described above, could be used as a primitive history mechanism. ESS provides a more sophisticated mechanism, however, which is described in Section 4.3 [Command History], page 18.

4.2.4 Keeping a record of your R session

To keep a record of your R session in a disk file, use the Emacs command *C-x C-w* (`write-file`) to attach a file to the ESS process buffer. The name of the process buffer will (probably) change to the name of the file, but this is not a problem. You can still use R as usual; just remember to save the file before you quit Emacs with *C-x C-s*. You can make ESS prompt you for a filename in which to save the transcript every time you start R by setting the variable `ess-ask-about-transfile` to `t`; See Section 3.4 [Customizing startup], page 14. For R files, naming transcript files '`*.Rout`' puts them in a special mode (ESS transcript mode — see Chapter 6 [Transcript Mode], page 26) for editing transcript files which is automatically selected for files with this suffix.

R transcripts can get very large, so some judicious editing is appropriate if you are saving it in a file. Use *C-c C-o* whenever a command produces excessively long output (printing large arrays, for example). Delete erroneous commands (and the resulting error messages or other output) by moving to the command (or its output) and typing *M-h C-w*. Also, remember that *C-c C-x* (and other hot keys) may be used for commands whose output you do not wish to appear in the transcript. These suggestions are appropriate even if you are not saving your transcript to disk, since the larger the transcript, the more memory your Emacs process will use on the host machine.

You can use `ess-transcript-clean-region` to strip output from a transcript, leaving only source code suitable for inclusion in files `source()`-able from R. see Section 6.2 [Clean], page 26,

4.3 Command History

ESS provides easy-to-use facilities for re-executing or editing previous commands. An input history of the last few commands is maintained (by default the last 500 commands are stored, although this can be changed by setting the variable `comint-input-ring-size` in `inferior-ess-mode-hook`.) The simplest history commands simply select the next and previous commands in the input history:

M-p `comint-previous-input` Select the previous command in the input history.

M-n `comint-next-input` Select the next command in the input history.

For example, pressing *M-p* once will re-enter the last command into the process buffer after the prompt but does not send it to the ESS process, thus allowing editing or correction of the command before the ESS process sees it. Once corrections have been made, press *RET* to send the edited command to the ESS process.

If you want to select a particular command from the history by matching it against a regular expression (see Section "Syntax of Regular Expression" in *The GNU Emacs Reference Manual*), to search for a particular variable name for example, these commands are also available:

M-r `comint-history-isearch-backward-regexp` Prompt for a regular expression, and search backwards through the input history for a command matching the expression.

A common type of search is to find the last command that began with a particular sequence of characters; the following two commands provide an easy way to do this:

C-c M-r `comint-previous-matching-input-from-input` Select the previous command in the history which matches the string typed so far.

C-c M-s `comint-next-matching-input-from-input` Select the next command in the history which matches the string typed so far.

Instead of prompting for a regular expression to match against, as they instead select commands starting with those characters already entered. For instance, if you wanted to re-execute the last `attach()` command, you may only need to type *att* and then *C-c M-r* and *RET*.

See Section "Shell History Ring" in *The GNU Emacs Reference Manual*, for a more detailed discussion of the history mechanism, and do experiment with the `In/Out` menu to explore the possibilities.

Many ESS users like to have even easier access to these, and recommend to add something like

```
(eval-after-load "comint"
 '(progn
    (define-key comint-mode-map [up]
      'comint-previous-matching-input-from-input)
    (define-key comint-mode-map [down]
      'comint-next-matching-input-from-input)

    ;; also recommended for ESS use --
```

```
(setq comint-move-point-for-output 'others)
;; somewhat extreme, almost disabling writing in *R*, *shell* buffers above prom
(setq comint-scroll-to-bottom-on-input 'this)
))
```

to your Emacs configuration file, where the last two settings are typically desirable for the situation where you work with a script (for example, `code.R`) and send code chunks to the process buffer (e.g. `*R*`). Note however that these settings influence all `comint`-using Emacs modes, not just the ESS ones, and for that reason, these customization cannot be part of `ESS` itself.

4.3.1 Saving the command history

The `ess-history-file` variable, which is `t` by default, together with `ess-history-directory`, governs if and where the command history is saved and restored between sessions. By default, `ess-history-directory` is `nil`, and the command history will be stored (as a file) in the same directory as the iESS process.

ESS users may work exclusively with script files rather than in a iESS session, and may not want to save any history files. To do so:

```
(setq ess-history-file nil)
```

or if you prefer only one global command history file:

```
(setq ess-history-directory "~/.R/")
```

in your Emacs configuration file.

4.4 References to historical commands

Instead of searching through the command history using the command described in the previous section, you can alternatively refer to a historical command directly using a notation very similar to that used in `csh`. History references are introduced by a '`!`' or '`^`' character and have meanings as follows:

'`!!`' The immediately previous command

'`!-`*N*' The *N*th previous command

'`!text`' The last command beginning with the string '`text`'

'`!?text`' The last command containing the string '`text`'

In addition, you may follow the reference with a *word designator* to select particular *words* of the input. A word is defined as a sequence of characters separated by whitespace. (You can modify this definition by setting the value of `comint-delimiter-argument-list` to a list of characters that are allowed to separate words and themselves form words.) Words are numbered beginning with zero. The word designator usually begins with a '`:`' (colon) character; however it may be omitted if the word reference begins with a '`^`', '`$`', '`*`' or '`-`'. If the word is to be selected from the previous command, the second '`!`' character can be omitted from the event specification. For instance, '`!!:1`' and '`!:1`' both refer to the first word of the previous command, while '`!!$`' and '`!$`' both refer to the last word in the previous command. The format of word designators is as follows:

'`0`' The zeroth word (i.e. the first one on the command line)

'*n*'	The *n*th word, where *n* is a number
'^'	The first word (i.e. the second one on the command line)
'$'	The last word
'*x*-*y*'	A range of words; '-*y*' abbreviates '0-*y*'
'*'	All the words except the zeroth word, or nothing if the command had just one word (the zeroth)
'*x**'	Abbreviates *x*-$
'*x*-'	Like '*x**', but omitting the last word

In addition, you may surround the entire reference except for the first '!' by braces to allow it to be followed by other (non-whitespace) characters (which will be appended to the expanded reference).

Finally, ESS also provides quick substitution; a reference like '^old^new^' means "the last command, but with the first occurrence of the string 'old' replaced with the string 'new'" (the last '^' is optional). Similarly, '^old^' means "the last command, with the first occurrence of the string 'old' deleted" (again, the last '^' is optional).

To convert a history reference as described above to an input suitable for R, you need to *expand* the history reference, using the TAB key. For this to work, the cursor must be preceded by a space (otherwise it would try to complete an object name) and not be within a string (otherwise it would try to complete a filename). So to expand the history reference, type *SPC TAB*. This will convert the history reference into an R command from the history, which you can then edit or press RET to execute.

For example, to execute the last command that referenced the variable data, type *!?data SPC TAB RET*.

4.5 Hot keys for common commands

ESS provides a number of commands for executing the commonly used functions. These commands below are basically information-gaining commands (such as objects() or search()) which tend to clutter up your transcript and for this reason some of the hot keys display their output in a temporary buffer instead of the process buffer by default. This behavior is controlled by the following option:

ess-execute-in-process-buffer [User Option]
: If non-nil, means that these commands will produce their output in the process buffer instead.

In any case, passing a prefix argument to the commands (with *C-u*) will reverse the meaning of ess-execute-in-process-buffer for that command. In other words, the output will be displayed in the process buffer if it usually goes to a temporary buffer, and vice-versa. These are the hot keys that behave in this way:

ess-execute-objects *posn* [Command]
: *C-c C-x* Sends the objects() command to the ESS process. A prefix argument specifies the position on the search list (use a negative argument to toggle ess-execute-in-process-buffer as well). This is a quick way to see what objects are in your

working directory. A prefix argument of 2 or more means get objects for that position. A negative prefix argument *posn* gets the objects for that position, as well as toggling `ess-execute-in-process-buffer`.

`ess-execute-search` *invert* [Command]
`C-c C-s` Sends the `inferior-ess-search-list-command` command to the `ess-language` process; `search()` in R. Prefix *invert* toggles `ess-execute-in-process-buffer`.

`ess-execute` may seem pointless when you could just type the command in anyway, but it proves useful for 'spot' calculations which would otherwise clutter your transcript, or for evaluating an expression while partway through entering a command. You can also use this command to generate new hot keys using the Emacs keyboard macro facilities; see Section "Keyboard Macros" in *The GNU Emacs Reference Manual*.

The following hot keys do not use `ess-execute-in-process-buffer` to decide where to display the output — they either always display in the process buffer or in a separate buffer, as indicated:

`ess-load-file` *filename* [Command]
`C-c M-l` Prompts for a file (*filename*) to load into the ESS process using `source()`. If there is an error during loading, you can jump to the error in the file with the following function.

`ess-parse-errors` *arg reset* [Command]
Visits next `next-error` message and corresponding source code. If all the error messages parsed so far have been processed already, the message buffer is checked for new ones. A prefix *arg* specifies how many error messages to move; negative means move back to previous error messages. Just `C-u` as a prefix means reparse the error message buffer and start at the first error. The *reset* argument specifies restarting from the beginning.

See Section 7.3 [Error Checking], page 28, for more details.

`ess-display-help-on-object` *object command* [Command]
`C-c C-v` Pops up a help buffer for an R object or function. If *command* is supplied, it is used instead of `inferior-ess-help-command`. See Chapter 8 [Help], page 34, for more details.

`ess-quit` [Command]
`C-c C-q` Issue an exiting command to the inferior process, additionally also running `ess-cleanup` for disposing of any temporary buffers (such as help buffers and edit buffers) that may have been created. Use this command when you have finished your R session instead of simply quitting at the inferior process prompt, otherwise you will need to issue the command `ess-cleanup` explicitly to make sure that all the files that need to be saved have been saved, and that all the temporary buffers have been killed.

4.6 Is the Statistical Process running under ESS?

For the R languages (R, S, S-Plus) ESS sets an option in the current process that programs in the language can check to determine the environment in which they are currently running.

ESS sets `options(STERM="iESS")` for R language processes running in an inferior `iESS[S]` or `iESS[R]` buffer.

ESS sets `options(STERM="ddeESS")` for independent S-Plus for Windows processes running in the GUI and communicating with ESS via the DDE (Microsoft Dynamic Data Exchange) protocol through a `ddeESS[S]` buffer.

Other values of `options()$STERM` that we recommend are:

- `length`: Fixed length xterm or telnet window.
- `scrollable`: Unlimited length xterm or telnet window.
- `server`: S-Plus Stat Server.
- `BATCH`: BATCH.
- `Rgui`: R GUI.
- `Commands`: S-Plus GUI without DDE interface to ESS.

Additional values may be recommended in the future as new interaction protocols are created. Unlike the values `iESS` and `ddeESS`, ESS can't set these other values since the R language program is not under the control of ESS.

4.7 Using emacsclient

When starting R or S under Unix, ESS sets `options(editor="emacsclient")`. Under Microsoft Windows, it will use gnuclient.exe rather than emacsclient, but the same principal applies. Within your R session, if you have a function called `iterator`, typing `fix(iterator)`, will show that function in a temporary Emacs buffer. You can then correct the function. When you kill the buffer, the definition of the function is updated. Using `edit()` rather than `fix()` means that the function is not updated. Finally, the R function `page(x)` will also show a text representation of the object `x` in a temporary Emacs buffer.

4.8 Other commands provided by inferior-ESS

The following commands are also available in the process buffer:

`comint-interrupt-subjob` [Command]
: *C-c C-c* Sends a Control-C signal to the iESS process. This has the effect of aborting the current command.

`ess-switch-to-inferior-or-script-buffer` *toggle-eob* [Command]
: *C-c C-z* When in process buffer, return to the most recent script buffer. When in a script buffer pop to the associated process buffer. Consecutive presses of *C-z* switch between the script and process buffers.

  If *toggle-eob* is given, the value of `ess-switch-to-end-of-proc-buffer` is toggled.

`ess-switch-to-end-of-proc-buffer` [User Option]
: If non-`nil`, `ess-switch-to-inferior-or-script-buffer` goes to end of process buffer.

Other commands available in iESS modes are discussed in Section “Shell Mode” in *The Gnu Emacs Reference Manual*.

5 Sending code to the ESS process

Other commands are also available for evaluating portions of code in the R process. These commands cause the selected code to be evaluated directly by the ESS process as if you had typed them in at the command line; the `source()` function is not used. You may choose whether both the commands and their output appear in the process buffer (as if you had typed in the commands yourself) or if the output alone is echoed. The behavior is controlled by the variable:

`ess-eval-visibly` [User Option]
: Non-`nil` means `ess-eval-*` commands display commands and output in the process buffer. Default is `t`.

Passing a prefix (*C-u*) (in elisp terms, the argument VIS) to any of the following commands, however, reverses the meaning of `ess-eval-visibly` for that command only — for example *C-u C-c C-j* evaluates the current line without showing the input in the iESS buffer. The default value of `ess-eval-visibly` (`t`) means that ESS calls block Emacs until they finish. This may be undesirable, especially if commands take long to finish. Users who want input to be displayed and Emacs not to be blocked can set `ess-eval-visibly` to `'nowait`. This sends the input to the iESS buffer but does not wait for the process to finish, ensuring Emacs is not blocked.

Primary commands for evaluating code are:

`ess-eval-region-or-line-and-step` *vis* [Command]
: *C-<RET>* Sends the highlighted region or current line and step to next line of code.

`ess-eval-region-or-function-or-paragraph` *vis* [Command]
: *C-M-x* Sends the current selected region or function or paragraph.

`ess-eval-region-or-function-or-paragraph-and-step` *vis* [Command]
: *C-c C-c* Like `ess-eval-region-or-function-or-paragraph` but steps to next line of code.

Other, not so often used, evaluation commands are:

`ess-eval-line` *vis* [Command]
: *C-c C-j* Sends the current line to the ESS process.

`ess-eval-line-and-go` *vis* [Command]
: *C-c M-j* Like `ess-eval-line` but additionally switches point to the ESS process.

`ess-eval-function` *vis no-error* [Command]
: *C-c C-f* Sends the function containing point to the ESS process.

`ess-eval-function-and-go` *vis* [Command]
: *C-c M-f* Like `ess-eval-function` but additionally switches point to the ESS process.

`ess-eval-region` *start end toggle message* [Command]
: *C-c C-r* Sends the current region to the ESS process.

`ess-eval-region-and-go` *start end vis* [Command]

C-c M-r Like `ess-eval-region` but additionally switches point to the ESS process.

`ess-eval-buffer` *vis* [Command]

C-c C-b Sends the current buffer to the ESS process.

`ess-eval-buffer-and-go` *vis* [Command]

C-c M-b Like `ess-eval-buffer` but additionally switches point to the ESS process.

All the above `ess-eval-*` commands are useful for evaluating small amounts of code and observing the results in the process buffer for debugging purposes, or for generating transcripts from source files. When editing R functions, it is generally preferable to use *C-c C-l* to update the function's value. In particular, `ess-eval-buffer` is now largely obsolete.

A useful way to work is to divide the frame into two windows; one containing the source code and the other containing the process buffer. If you wish to make the process buffer scroll automatically when the output reaches the bottom of the window, you will need to set the variable `comint-move-point-for-output` to `'others` or `t`.

6 Manipulating saved transcript files

Inferior R mode records the transcript (the list of all commands executed, and their output) in the process buffer, which can be saved as a *transcript file*, which should normally have the suffix `.Rout`. The most obvious use for a transcript file is as a static record of the actions you have performed in a particular R session. Sometimes, however, you may wish to re-execute commands recorded in the transcript file by submitting them to a running ESS process. This is what Transcript Mode is for.

If you load file a with the suffix `.Rout` into Emacs, it is placed in R Transcript Mode. Transcript Mode is similar to inferior R mode (see Chapter 4 [Entering commands], page 15): paragraphs are defined as a command and its output, and you can move though commands either with the paragraph commands or with *C-c C-p* and *C-c C-n*.

6.1 Resubmitting commands from the transcript file

Three commands are provided to re-submit command lines from the transcript file to a running ESS process. They are:

`ess-transcript-send-command` [Command]
: *M-RET* Sends the current command line to the ESS process, and execute it.

`ess-transcript-copy-command` [Command]
: *C-c RET* Copy the current command to the ESS process, and switch to it (ready to edit the copied command).

`ess-transcript-send-command-and-move` [Command]
: *RET* Sends the current command to the ESS process, and move to the next command line. This command is useful for submitting a series of commands.

Note that the first two commands are similar to those on the same keys in inferior R Mode. In all three cases, the commands should be executed when the cursor is on a command line in the transcript; the prompt is automatically removed before the command is submitted.

6.2 Cleaning transcript files

Yet another use for transcript files is to extract the command lines for inclusion in an R source file or function. Transcript mode provides one command which does just this:

`ess-transcript-clean-region` *beg end even-if-read-only* [Command]
: *C-c C-w* Strip the transcript in the region (given by *beg* and *end*), leaving only commands. Deletes any lines not beginning with a prompt, and then removes the prompt from those lines that remain. Prefix argument *even-if-read-only* means to clean even if the buffer is read-only. Don't forget to remove any erroneous commands first!

The remaining command lines may then be copied to a source file or edit buffer for inclusion in a function definition, or may be evaluated directly (see Chapter 5 [Evaluating code], page 24) using the code evaluation commands from R mode, also available in R Transcript Mode.

7 Editing objects and functions

ESS provides facilities for editing R objects within your Emacs session. Most editing is performed on R functions, although in theory you may edit datasets as well. Edit buffers are always associated with files, although you may choose to make these files temporary if you wish. Alternatively, you may make use of a simple yet powerful mechanism for maintaining backups of text representations of R functions. Error-checking is performed when R code is loaded into the ESS process.

7.1 Creating or modifying R objects

To edit an object, type

`ess-dump-object-into-edit-buffer` *object* [Command]
C-c C-e C-d Edit an object in its own edit buffer.

from within the iESS process buffer (`*R*`). You will then be prompted for an object to edit: you may either type in the name of an existing object (for which completion is available using the *TAB* key), or you may enter the name of a new object. A buffer will be created containing the text representation of the requested object or, if you entered the name of a non-existent object at the prompt and the variable `ess-function-template` is non-`nil`, you will be presented with a template defined by that variable, which defaults to a skeleton function construct.

You may then edit the function as required. The edit buffer generated by `ess-dump-object-into-edit-buffer` is placed in the `ESS` major mode which provides a number of commands to facilitate editing R source code. Commands are provided to intelligently indent R code, evaluate portions of R code and to move around R code constructs.

Note: when you dump a file with *C-c C-e C-d*, ESS first checks to see whether there already exists an edit buffer containing that object and, if so, pops you directly to that buffer. If not, ESS next checks whether there is a file in the appropriate place with the appropriate name (see Section 7.6 [Source Files], page 30) and if so, reads in that file. You can use this facility to return to an object you were editing in a previous session (and which possibly was never loaded to the R session). Finally, if both these tests fail, the ESS process is consulted and a `dump()` command issued. If you want to force ESS to ask the ESS process for the object's definition (say, to reformat an unmodified buffer or to revert back to R's idea of the object's definition) pass a prefix argument to `ess-dump-object-into-edit-buffer` by typing *C-u C-c C-e C-d*.

7.2 Loading source files into the ESS process

The best way to get information — particularly function definitions — into R is to load them in as source file, using R's `source` function. You have already seen how to create source files using *C-c C-e C-d*; ESS provides a complementary command for loading source files (even files not created with ESS!) into the ESS process, namely `ess-load-file` (*C-c M-l*). see Section 4.5 [Hot keys], page 20.

After typing *C-c M-l* you will prompt for the name of the file to load into R; usually this is the current buffer's file which is the default value (selected by simply pressing *RET* at the

prompt). You will be asked to save the buffer first if it has been modified (this happens automatically if the buffer was generated with *C-c C-e C-d*). The file will then be loaded, and if it loads successfully you will be returned to the ESS process.

7.3 Detecting errors in source files

If any errors occur when loading a file with `C-c C-l`, ESS will inform you of this fact. In this case, you can jump directly to the line in the source file which caused the error by typing *C-c ‘* (`ess-parse-errors`). You will be returned to the offending file (loading it into a buffer if necessary) with point at the line S reported as containing the error. You may then correct the error, and reload the file. Note that none of the commands in an R source file will take effect if any part of the file contains errors.

Sometimes the error is not caused by a syntax error (loading a non-existent file for example). In this case typing *C-c ‘* will simply display a buffer containing S's error message. You can force this behavior (and avoid jumping to the file when there *is* a syntax error) by passing a prefix argument to `ess-parse-errors` with *C-u C-c ‘*.

7.4 Indenting and formatting R code

ESS provides a sophisticated mechanism for indenting R source code. Compound statements (delimited by ‘`{`’ and ‘`}`’) are indented relative to their enclosing block. In addition, the braces have been electrified to automatically indent to the correct position when inserted, and optionally insert a newline at the appropriate place as well. Lines which continue an incomplete expression are indented relative to the first line of the expression. Function definitions, `if` statements, calls to `expression()` and loop constructs are all recognized and indented appropriately. User variables are provided to control the amount of indentation in each case, and there are also a number of predefined indentation styles to choose from.

Comments are also handled specially by ESS, using an idea borrowed from the Emacs-Lisp indentation style. By default, comments beginning with ‘`###`’ are aligned to the beginning of the line. Comments beginning with ‘`##`’ are aligned to the current level of indentation for the block containing the comment. Finally, comments beginning with ‘`#`’ are aligned to a column on the right (the 40th column by default, but this value is controlled by the variable `comment-column`,) or just after the expression on the line containing the comment if it extends beyond the indentation column. You turn off the default behavior by adding the line `(setq ess-indent-with-fancy-comments nil)` to your `.emacs` file.

ESS also supports Roxygen entries which is R documentation maintained in the source code as comments See Section 10.2.2 [Roxygen], page 42.

The indentation commands provided by ESS are:

`ess-indent-or-complete` [Command]

TAB Indents the current line as R code.

Try to indent first, and if code is already properly indented, complete instead. In ess-mode, only tries completion if `ess-tab-complete-in-script` is non-nil. See also `ess-first-tab-never-complete`.

`ess-tab-complete-in-script` [User Option]

If non-`nil`, *TAB* in script buffers tries to complete if there is nothing to indent.

`ess-first-tab-never-complete` [User Option]
: If non-`nil`, *TAB* never tries to complete in ess-mode. The default `'symbol` does not try to complete if the next char is a valid symbol constituent. There are more options, see the help (*C-h v*).

`ess-indent-exp` [Command]
: *TAB* Indents each line in the R (compound) expression which follows point. Very useful for beautifying your R code.

`ess-electric-brace` [Command]
: *{ }* The braces automatically indent to the correct position when typed.

The following Emacs command are also helpful:

RET
LFD `newline-and-indent` Insert a newline, and indent the next line. (Note that most keyboards nowadays do not have a `LINEFEED` key, but `RET` and *C-j* are equivalent.)

M-; `indent-dwim` Call the comment command you want (Do What I Mean).

7.4.1 Changing styles for code indentation and alignment

The combined value of twelve variables (4 of three groups `ess-indent-*`, `ess-offset-*` and `ess-align-*`) that control indentation are collectively termed a *style*. ESS provides several styles covering the common styles of indentation: `DEFAULT`, `OWN`, `GNU`, `BSD`, `K&R`, `C++`, `RRR`, `RRR+`, `Rstudio`, `Rstudio-`, and `CLB`. The variable `ess-style-alist` lists the value of each indentation variable per style (and its documentation contains more).

`ess-set-style` [Command]
: *C-c C-e C-s* (or *C-c C-e s*) sets the formatting style in this buffer to be one of the predefined styles, see above. The `DEFAULT` style uses the default values for the indenting variables; The `OWN` style allows you to use your own private values of the indentation variable, see below.

`ess-default-style` [User Option]
: The default value of `ess-style`. See the variable `ess-style-alist` for how these groups (DEFAULT, OWN, GNU, RRR, . . .) map onto different settings for variables. You can set this in your Emacs configuration file:

```
(setq ess-default-style 'C++)
```

`ess-style-alist` [User Option]
: Predefined formatting styles for ESS code. Values for all groups, except OWN, are fixed. To change the value of variables in the OWN group, customize the variable `ess-own-style-list`. The default style in use is controlled by `ess-default-style`.

The styles `DEFAULT` and `OWN` are initially identical. If you wish to edit some of the default values, set `ess-default-style` to `'OWN` and change `ess-own-style-list`. See Appendix A [Customization], page 68, for convenient ways to set both these variables.

If you prefer not to use the customization facility, you can change individual indentation variables within a hook, for example:

```
(defun myindent-ess-hook ()
```

```
  (setq ess-indent-level 4))
(add-hook 'ess-mode-hook 'myindent-ess-hook)
```

In the rare case that you'd like to add an entire new indentation style of your own, copy the definition of `ess-own-style-list` to a new variable and ensure that the last line of the `:set` declaration calls `ess-add-style` with a unique name for your style (e.g. `'MINE`). Finally, add `(setq ess-default-style 'MINE)` to use your new style.

7.5 Commands for motion, completion and more

A number of commands are provided to move across function definitions in the edit buffer:

`ess-goto-beginning-of-function-or-para` [Command]
: *ESC* `C-a` aka *C-M-a* If inside a function go to the beginning of it, otherwise go to the beginning of paragraph.

`ess-goto-end-of-function-or-para` [Command]
: *ESC* `C-e` aka *C-M-e* Move point to the end of the function containing point.

`ess-mark-function` [Command]
: *ESC* `C-h` aka *C-M-h* Place point at the beginning of the R function containing point, and mark at the end.

Don't forget the usual Emacs commands for moving over balanced expressions and parentheses: See Section "Lists and Sexps" in *The GNU Emacs Reference Manual*.

Completion is provided in the edit buffer in a similar fashion to the process buffer: `TAB` first indents, and if there is nothing to indent, completes the object or file name; *M-?* lists file completions. See See Chapter 9 [Completion], page 36, for more.

Finally, *C-c C-z* (`ess-switch-to-inferior-or-script-buffer`) returns you to the `iESS` process buffer, if done from a script buffer, placing point at the end of the buffer. If this is done from the `iESS` process buffer, point is taken to the script buffer.

In addition many commands available in the process buffer are also available in the script buffer. You can still read help files with *C-c C-v*, edit another function with *C-c C-e C-d* and of course *C-c C-l* can be used to load a source file into R.

7.6 Maintaining R source files

Every edit buffer in ESS is associated with a *dump file* on disk. Dump files are created whenever you type *C-c C-e C-d* (`ess-dump-object-into-edit-buffer`), and may either be deleted after use, or kept as a backup file or as a means of keeping several versions of an R function.

`ess-delete-dump-files` [User Option]
: If non-`nil`, dump files created with C-c C-e C-d are deleted immediately after they are created by the ess-process.

Since immediately after R dumps an object's definition to a disk file the source code on disk corresponds exactly to R's idea of the object's definition, the disk file isn't needed; deleting it now has the advantage that if you *don't* modify the file (say, because you just wanted to look at the definition of one of the standard R functions) the source dump file

won't be left around when you kill the buffer. Note that this variable only applies to files generated with R's `dump` function; it doesn't apply to source files which already exist. The default value is `t`.

`ess-keep-dump-files` [User Option]
: Variable controlling whether to delete dump files after a successful load. If '`nil`': always delete. If '`ask`', confirm to delete. If '`check`', confirm to delete, except for files created with `ess-dump-object-into-edit-buffer`. Anything else, never delete. This variable only affects the behaviour of `ess-load-file`. Dump files are never deleted if an error occurs during the load.

After an object has been successfully (without error) loaded back into R with `C-c C-l`, the disk file again corresponds exactly (well, almost — see below) to R's record of the object's definition, and so some people prefer to delete the disk file rather than unnecessarily use up space. This option allows you to do just that.

If the value of `ess-keep-dump-files` is `t`, dump files are never deleted after they are loaded. Thus you can maintain a complete text record of the functions you have edited within ESS. Backup files are kept as usual, and so by using the Emacs numbered backup facility — see Section "Single or Numbered Backups" in *The Gnu Emacs Reference Manual*, you can keep a historic record of function definitions. Another possibility is to maintain the files with a version-control system such as git See Section "Version Control" in *The Gnu Emacs Reference Manual*. As long as a dump file exists in the appropriate place for a particular object, editing that object with `C-c C-e C-d` finds that file for editing (unless a prefix argument is given) — the ESS process is not consulted. Thus you can keep comments *outside* the function definition as a means of documentation that does not clutter the R object itself. Another useful feature is that you may format the code in any fashion you please without R re-indenting the code every time you edit it. These features are particularly useful for project-based work.

If the value of `ess-keep-dump-files` is nil, the dump file is always silently deleted after a successful load with `C-c C-l`. While this is useful for files that were created with `C-c C-e C-d` it also applies to any other file you load (say, a source file of function definitions), and so can be dangerous to use unless you are careful. Note that since `ess-keep-dump-files` is buffer-local, you can make sure particular files are not deleted by setting it to `t` in the Local Variables section of the file See Section "Local Variables in Files" in *The Gnu Emacs Reference Manual*.

A safer option is to set `ess-keep-dump-files` to `ask`; this means that ESS will always ask for confirmation before deleting the file. Since this can get annoying if you always want to delete dump files created with `C-c C-e C-d`, but not any other files, setting `ess-keep-dump-files` to `check` (the default value) will silently delete dump files created with `C-c C-e C-d` in the current Emacs session, but query for any other file. Note that in any case you will only be asked for confirmation once per file, and your answer is remembered for the rest of the Emacs session.

Note that in all cases, if an error (such as a syntax error) is detected while loading the file with `C-c C-l`, the dump file is *never* deleted. This is so that you can edit the file in a new Emacs session if you happen to quit Emacs before correcting the error.

Dump buffers are always autosaved, regardless of the value of `ess-keep-dump-files`.

7.7 Names and locations of dump files

Every dump file should be given a unique file name, usually the dumped object name with some additions.

`ess-dump-filename-template` [User Option]
Template for filenames of dumped objects. `%s` is replaced by the object name.

By default, dump file names are the user name, followed by '`.`' and the object and ending with '`.R`'. Thus if user `joe` dumps the object `myfun` the dump file will have name `joe.myfun.R`. The username part is included to avoid clashes when dumping into a publicly-writable directory, such as `/tmp`; you may wish to remove this part if you are dumping into a directory owned by you.

You may also specify the directory in which dump files are written:

`ess-source-directory` [User Option]
Directory name (ending in a slash) where R dump files are to be written.

By default, dump files are always written to `/tmp`, which is fine when `ess-keep-dump-files` is `nil`. If you are keeping dump files, then you will probably want to keep them somewhere in your home directory, say `~/R-source`. This could be achieved by including the following line in your Emacs configuration file:

```
(setq ess-source-directory (expand-file-name "~/R-source/"))
```

If you would prefer to keep your dump files in separate directories depending on the value of some variable, ESS provides a facility for this also. By setting `ess-source-directory` to a lambda expression which evaluates to a directory name, you have a great deal of flexibility in selecting the directory for a particular source file to appear in. The lambda expression is evaluated with the process buffer as the current buffer and so you can use the variables local to that buffer to make your choice. For example, the following expression causes source files to be saved in the subdirectory `Src` of the directory the ESS process was run in.

```
(setq ess-source-directory
      (lambda ()
        (concat ess-directory "Src/")))
```

(`ess-directory` is a buffer-local variable in process buffers which records the directory the ESS process was run from.) This is useful if you keep your dump files and you often edit objects with the same name in different ESS processes. Alternatively, if you often change your R working directory during an R session, you may like to keep dump files in some subdirectory of the directory pointed to by the first element of the current search list. This way you can edit objects of the same name in different directories during the one R session:

```
(setq ess-source-directory
   (lambda ()
       (file-name-as-directory
        (expand-file-name (concat
                           (car ess-search-list)
                           "/.Src")))))
```

If the directory generated by the lambda function does not exist but can be created, you will be asked whether you wish to create the directory. If you choose not to, or the directory cannot be created, you will not be able to edit functions.

8 Reading help files

ESS provides an easy-to-use facility for reading R help files from within Emacs. From within the ESS process buffer or any ESS edit buffer, typing *C-c C-v* (`ess-display-help-on-object`) will prompt you for the name of an object for which you would like documentation. Completion is provided over all objects which have help files.

If the requested object has documentation, you will be popped into a buffer (named `*help(`*obj-name*`)*`) containing the help file. This buffer is placed in a special ESS help mode which disables the usual editing commands but which provides a number of keys for paging through the help file.

Help commands:

? `ess-describe-help-mode` Pops up a help buffer with a list of the commands available in R help mode.

h `ess-display-help-on-object` Pop up a help buffer for a different object.

Paging commands:

DEL `scroll-down-command` Move one screen backwards through the help file.

SPC `scroll-up-command` Move one screen forwards through the help file.

>
< `end/beginning-of-buffer` Move to the end or beginning of the help file, respectively.

Section-based motion commands:

n
p `ess-skip-to-previous-section` and `ess-skip-to-next-section` Move to the next and previous section header in the help file, respectively. A section header consists of a number of capitalized words, followed by a colon.

In addition, the *s* key followed by one of the following letters will jump to a particular section in the help file. Note that the exact headings available and capitalization scheme may vary across languages.

You may use *s ?* to get the current list of active key bindings.

'a' Arguments:

'b' Background:

'B' Bugs:

'd' Description:

'D' Details:

'e' Examples:

'n' Note:

'O' Optional Arguments:

'R' Required Arguments:

'r' References:

's' See Also:

'S' Side Effects:

'u' Usage:

'v' Value:

'?' Pops up a help buffer with a list of the defined section motion keys.

Evaluation:

l `ess-eval-line-and-step` Evaluates the current line in the ESS process, and moves to the next line. Useful for running examples in help files.

r `ess-eval-region` Send the contents of the current region to the ESS process. Useful for running examples in help files.

Quit commands:

q `ess-help-quit` Return to previously selected buffer, and bury the help buffer.

k `kill-buffer` Return to previously selected buffer, and kills the help buffer.

x `ess-kill-buffer-and-go` Return to the ESS process, killing this help buffer.

Miscellaneous:

i `ess-display-index` Prompt for a package and display it's help index.

v `ess-display-vignettes` Display all available vignettes.

w `ess-display-help-in-browser` Display current help page with the web browser.

/ `isearch-forward` Same as *C-s*.

In addition, all of the ESS commands available in the edit buffers are also available in R help mode (see Section 7.1 [Edit buffer], page 27). Of course, the usual (non-editing) Emacs commands are available, and for convenience the digits and `-` act as prefix arguments.

If a help buffer already exists for an object for which help is requested, that buffer is popped to immediately; the ESS process is not consulted at all. If the contents of the help file have changed, you either need to kill the help buffer first, or pass a prefix argument (with *C-u*) to `ess-display-help-on-object`.

Help buffers are marked as temporary buffers in ESS, and are deleted when `ess-quit` or `ess-cleanup` are called.

Help buffers normally appear in another window within the current frame. If you wish help buffers to appear in their own frame (either one per help buffer, or one for all help buffers), you can customize the variable `ess-help-own-frame`.

9 Completion

9.1 Completion of object names

The `TAB` key is for completion. The value of the variable `ess-first-tab-never-complete` controls when completion is allowed controls when completion is allowed to occur. In `ess-mode` `TAB` first tries to indent, and if there is nothing to indent, complete the object name instead.

`TAB` `comint-dynamic-complete` Complete the R object name or filename before point.

When the cursor is just after a partially-completed object name, pressing `TAB` provides completion in a similar fashion to the rest of Emacs. ESS maintains a list of all objects known to R at any given time, which basically consists of all objects (functions and datasets) in every attached directory listed by the `search()` command along with the component objects of attached data frames

For example, consider three functions `binomplot()`, `binom.test()` and `binomial()`. Typing *`bin TAB`* will insert the characters '`om`', completing the longest prefix ('`binom`') which distinguishes these three commands. Pressing *`TAB`* once more provides a list of the three commands which have this prefix, allowing you to add more characters (say, '`.`') which specify the function you desire. After entering more characters pressing *`TAB`* yet again will complete the object name up to uniqueness, etc. If you just wish to see what completions exist without adding any extra characters, type *`M-?`*.

`ess-list-object-completions` [Command]
`M-?` List all possible completions of the object name at point.

ESS also provides completion over the components of named lists and environments (after '`$`'), S4 classes slots (after @), package and namespace objects (after :: and :::).

Completion is also provided over file names, which is particularly useful when using R functions such as `get()` or `scan()` which require fully expanded file names.

In the iESS buffer, if the cursor is not in a string and does not follow a (partial) object name, the `TAB` key has a third use: it expands history references. See Section 4.4 [History expansion], page 19.

Efficiency in completion is gained by maintaining a cache of objects currently known to R; when a new object becomes available or is deleted, only one component of the cache corresponding to the associated directory needs to be refreshed. If ESS ever becomes confused about what objects are available for completion (such as when if refuses to complete an object you **know** is there), the command *`M-x ess-resynch`* forces the *entire* cache to be refreshed, which should fix the problem.

9.2 Completion of function arguments

When inside a function call (i.e. following '`(`'), *`TAB`* completion also provides function arguments. If function is a generic, completion will provide all the arguments of `S3` methods known to `R`.

A related functionality is See Section 11.1 [ESS ElDoc], page 45, which displays function arguments in the echo area whenever the point is inside a function call.

9.3 Minibuffer completion

ESS uses IDO mechanism (part of default Emacs) for minibuffer completion if the `ido` package is available and the value of `ess-use-ido` it `t` (the default). The completion command `ess-completing-read` falls back on classic `completion-read` interface if this feature is not available for whatever reason.

9.4 Integration with auto-complete package

ESS provides three sources for `Auto Completion` mode: `ac-source-R-args`, `ac-source-R-objects` and `ac-source-R`. The last one combines the previous two and makes them play nicely together. See auto-complete package documentation (`http://cx4a.org/software/auto-complete/`) for how to modify and install your own completion sources.

For the default auto-complete ESS configuration, install the latest version of `auto-complete` package. ESS automatically detect the package and activates auto-complete in ESS buffers.

To deactivate AC, place the following into your init file:

```
(setq ess-use-auto-complete nil)
```

Or, to activate auto-completion only in the `ess-mode` buffers:

```
(setq ess-use-auto-complete 'script-only)
```

ESS provides AC help both for arguments and objects (default keys *C-?* or *<f1>*). You can bind *M-h* to display quick help pop-ups:

```
(define-key ac-completing-map (kbd "M-h") 'ac-quick-help)
```

`AC` binds *M-n*, and *M-p* for the navigation in the completion menu, which might be inconvenient if you want it to use in the inferior R. Bind it to something else:

```
(define-key ac-completing-map "\M-n" nil) ;; was ac-next
(define-key ac-completing-map "\M-p" nil) ;; was ac-previous
(define-key ac-completing-map "\M-," 'ac-next)
(define-key ac-completing-map "\M-k" 'ac-previous)
```

9.5 Company

Another popular package for completion is company, short for complete anything. ESS provides support for company out-of-the-box. To disable company support, set `ess-use-company` to `nil`. You can set it to `'script-only` to only activate company in R scripts.

9.6 Icicles

Another option for comprehensively handling completion in Emacs is via Icicles (`https://www.emacswiki.org/emacs/Icicles`). It allows users to have completions shown temporarily in the standard '`*Completions*`' buffer, and interactively select completion candidates using several methods. As of version 2013.04.04, Icicles provides support for completion in ESS. Please consult Icicles documentation, which is easily accessible via *customize-group Icicles*, for more details on installation and customization options.

Once installed, Icicles can be activated by evaluating):

```
(require 'icicles)
```

```
(icy-mode 1)
```

Icicles can be toggled by typing *M-x icy*.

When Icicles is on, *TAB* offers completion, provided the conditions determined by `ess-first-tab-never-complete` allow it. Typing *M-TAB* will attempt completion regardless. Typing *M-?* in ESS or iESS modes brings up the relevant completion candidates from which to choose. Minibuffer input filters the available candidates. Use *TAB* for prefix completion or *S-TAB* for substring or regexp completion. Use *S-SPC* to match an additional pattern (repeatable). You can cycle among the matching candidates, choosing with *RET* or *mouse-2*.

10 Developing with ESS

ESS provides several tools to help you with the development of your R packages:

10.1 ESS tracebug

ESS `tracebug` offers visual debugging, interactive error navigation, interactive backtrace, breakpoint manipulation, control over R error actions, watch window and interactive flagging/unflagging of functions for debugging.

`ess-tracebug` is on by default. You can toggle it on and off with *M-x* `ess-tracebug`. To disable startup activation of `ess-tracebug` set `ess-use-tracebug` to nil.

Tracebug functionality can be found on `ess-dev-map`, bound to *C-c C-t*. Additionally, when subprocess is in a debugging state `ess-debug-minor-mode` is active and the following additional shortcuts are available:

```
* Interactive Debugging (‘ess-debug-minor-mode-map’):

 M-C   . Continue                  . ‘ess-debug-command-continue’
 M-C-C . Continue multi            . ‘ess-debug-command-continue-multi’
 M-N   . Next step                 . ‘ess-debug-command-next’
 M-C-N . Next step multi           . ‘ess-debug-command-next-multi’
 M-U   . Up frame                  . ‘ess-debug-command-up’
 M-Q   . Quit debugging            . ‘ess-debug-command-quit’
```

These are all the tracebug commands defined in `ess-dev-map` (*C-c C-t ?* to show this table):

```
* Breakpoints (‘ess-dev-map’):

 b   . Set BP (repeat to cycle BP type) . ‘ess-bp-set’
 B   . Set conditional BP               . ‘ess-bp-set-conditional’
 k   . Kill BP                          . ‘ess-bp-kil’
 K   . Kill all BPs                     . ‘ess-bp-kill-all’
 o   . Toggle BP state                  . ‘ess-bp-toggle-state’
 l   . Set logger BP                    . ‘ess-bp-set-logger’
 n   . Goto next BP                     . ‘ess-bp-next’
 p   . Goto previous BP                 . ‘ess-bp-previous’

  (C- prefixed equivalents are also defined)

* Debugging (‘ess-dev-map’):
 ‘   . Show traceback                      . ‘ess-show-traceback’ (also on C-c ‘)
 ~   . Show callstack                      . ‘ess-show-call-stack’ (also on C-c ~)
 e   . Toggle error action (repeat to cycle). ‘ess-debug-toggle-error-action’
 d   . Flag for debugging                  . ‘ess-debug-flag-for-debugging’
 u   . Unflag for debugging                . ‘ess-debug-unflag-for-debugging’
 w   . Watch window                        . ‘ess-watch’
```

```
  (C- prefixed equivalents are also defined)

* Navigation to errors (general Emacs functionality):

 C-x `, M-g n    . `next-error'
 M-g p           . `previous-error'

* Misc:

?   . Show this help             . `ess-tracebug-show-help'
```

To configure how electric watch window splits the display see `ess-watch-width-threshold` and `ess-watch-height-threshold` variables.

Note: Currently, ess-tracebug does not detect some of R's debug related messages in non-English locales. To set your R messages to English add the following line to your .Rprofile init file:

```
Sys.setlocale("LC_MESSAGES", "C")
```

10.1.1 Getting started with tracebug

Consider a buffer with the following function:

```
test <- function(x){
  mean(x),
}
```

Evaluating the function (e.g. `C-c C-c`) results in an error about an unexpected comma. You can use `next-error` (bound by default to `C-x `, `M-g n`, and `M-g M-n`) to jump to the place where the error occurred. Alternatively, use the mouse to click on the error to jump to where it occurred.

Correct the error by deleting the comma. Now put point on `mean` and set the breakpoint using *`C-c C-t b`* (`ess-bp-set`) and reevaluate the function. Jump to the inferior buffer (possibly using `C-c C-z`) and evaluate `test(1:10)`. An interactive debug process starts, stopping at the breakpoint we just specified. Here you can debug your function (what is `x` at this point?). Use `M-N` to continue.

Let's replace our test function with one slightly more complicated:

```
test <- function(x){
  x <- x + 1
  y <- mean(x)
  x <- ifelse(x > 5, x, x - 100)
  list(x, y)
}
```

Try setting multiple breakpoints. You can unset a breakpoint by killing it with *`C-c C-t k`*. You can set conditional breakpoints too. Try setting one by placing point on the line `x <- x + 1` and doing *`C-c C-t B`*. ESS will ask for the condition. Let's set it to `!is.numeric(x)`. After re-evaluating `test`, try calling `test(1:100)` and `test('foo')`.

You can remove all breakpoints with *`C-c C-t K`*.

You can flag a function for debugging (similar to calling `debug(test)` at R's prompt) by doing *C-c C-t d*. Try this yourself by putting point over `test` and doing *C-c C-t d*.

If an error occurs, you can get the complete call stack by doing *C-c ‘* or *C-c C-t ‘* (`ess-show-traceback`).

Tracebug also offers a watch window where you can watch values of objects. Open it with *C-c C-t w* (`ess-watch`). Initially you aren't watching anything. Add something with `a` (e.g. `a test`). The watch window displays what the object is at any given time and automatically updates. Quit the watch window with *q*.

10.2 Editing documentation

ESS provides two ways of writing documentation for R objects. Either using the standard R documentation system or using in-source documentation written as structured comment fields for use with the Roxygen package.

10.2.1 Editing R documentation (Rd) files

R objects are documented in files written in the *R documentation* ("Rd"), a simple markup language closely resembling (La)TEX, which can be processed into a variety of formats, including LaTEX, HTML, and plain text. Rd format is described in section "Rd format" of the "Writing R Extensions" manual in the R distribution. ESS has several features that facilitate editing Rd files.

Visiting an Rd file as characterized by its extension `Rd` will activate Rd Mode, which provides several facilities for making editing R documentation files more convenient, by helping with indentation, insertions, even doing some of the typing for you (with Abbrev Mode), and by showing Rd keywords, strings, etc. in different faces (with Font Lock Mode).

Note that R also accepts Rd files with extension `rd`; to activate ESS[Rd] support for this extension, you may need to add

```
(add-to-list 'auto-mode-alist '("\\.rd\\'" . Rd-mode))
```

to one of your Emacs startup files.

In Rd mode, the following special Emacs commands can be used in addition to the standard Emacs commands.

C-h m Describe the features of Rd mode.

LFD
RET Reindent the current line, insert a newline and indent the new line (`reindent-then-newline-and-indent`). An abbrev before point is expanded if `abbrev-mode` is non-`nil`.

TAB Indent current line based on its contents and on previous lines. (`indent-according-to-mode`).

C-c C-e Insert a "skeleton" with Rd markup for at least all mandatory entries in Rd files (`Rd-mode-insert-skeleton`). Note that many users might prefer to use the R function `prompt` on an existing R object to generate a non-empty Rd "shell" documenting the object (which already has all information filled in which can be obtained from the object).

C-c C-f Insert "font" specifiers for some of the Rd markup commands markup available for emphasizing or quoting text, including markup for URLs and email addresses (`Rd-font`). *C-c C-f* is only a prefix; see e.g. *C-c C-f TAB* for the available bindings. Note that currently, not all of the Rd text markup as described in section "Marking text" of "Writing R Extensions" can be accessed via *C-c C-f*.

C-c C-j Insert a suitably indented '`\item{`' on the next line (`Rd-mode-insert-item`).

C-c C-p Preview a plain text version ("help file", see Chapter 8 [Help], page 34) generated from the Rd file (`Rd-preview-help`).

In addition, when editing Rd files one can interact with a running R process in a similar way as when editing R language files. For example, *C-c C-v* provides access to on-line help, and *C-c C-n* sends the current line to the R process for evaluation. This interaction is particularly useful when editing the examples in the Rd file. See *C-h m* for all available commands.

Rd mode also provides access to abbreviations for most of the Rd markup commands. Type *M-x list-abbrevs* with Abbrev mode turned on to list all available abbrevs. Note that all Rd abbrevs start with a grave accent.

Rd mode can be customized via the following variables.

`Rd-mode-hook` [User Option]
Hook to be run when Rd mode is entered.

`Rd-indent-level` [User Option]
The indentation of Rd code with respect to containing blocks. Default is 2.

`Rd-to-help-command` [User Option]
The shell command used for converting Rd source to help text. Default is '`R CMD Rd2txt`'.

To automatically turn on the abbrev and font-lock features of Rd mode, add the following lines to one of your Emacs startup files:

```
(add-hook 'Rd-mode-hook
          (lambda ()
           (abbrev-mode 1)
           (font-lock-mode 1)))
```

10.2.2 Editing Roxygen documentation

The Roxygen R package makes it possible to keep the intended contents for Rd files as structured comments in the R source files. Roxygen can then parse R files and generate appropriate Rd files from these comments, fill the usage fields as well as updating `NAMESPACE` files. See the Roxygen documentation found via `http://roxygen.org` for more information on its usage. An example of an Roxygen entry for a simple R function can look like this:

```
##' Description of the function
##'
##' Further details about this function
##' @title A title
##' @param me all parameters must be listed and documented
```

```
##' @return Description of the return value
##' @author The author
myfun <- function(me)
  cat("Hello", me, "\n")
```

The entry is immediately preceding the object to document and all lines start with the Roxygen prefix string, in this case `##'`. ESS provides support to edit these documentation entries by providing line filling, navigation, template generation etc. Syntax highlighting is provided for Emacs but not for XEmacs.

Roxygen is customized by the variables in the customization group "Ess Roxy". Customizables include the Roxygen prefix, use of folding to toggle visibility of Roxygen entries and the Roxygen template.

All ESS Roxygen support is defined in `ess-roxy.el` which is loaded by default in ESS. The following special Emacs commands are provided.

`ess-roxy-update-entry` [Command]

C-c C-o C-o Generate a Roxygen template or update the parameter list in Roxygen entry at point (or above the function at the point). Documented parameters that are not in the function are placed last in the list, parameters that are not documented and not in the definition are dropped. Parameter descriptions are filled if `ess-roxy-fill-param-p` is non-nil.

`ess-roxy-toggle-roxy-region` *beg end* [Command]

C-c C-o C-c Toggle the presence of the leading Roxygen string on all lines in the marked region (between *beg* and *end*. Convenient for transferring text to Roxygen entries and to evaluate example fields.

`ess-roxy-preview-Rd` *name-file* [Command]

C-c C-o C-r Use the attached R process to parse the entry at point to obtain the Rd code. Convenient for previewing and checking syntax. When used with the prefix argument *name-file*, i.e. *C-u C-c C-e C-r*, place the content in a buffer associated with a Rd file with the same name as the documentation. Requires the Roxygen package to be installed.

`ess-roxy-preview-HTML` *visit-instead-of-open* [Command]

C-c C-o C-t Use the attached R process to parse the entry at to generate HTML for the entry and open it in a browser. When used with the prefix argument *visit-instead-of-open*, i.e. *C-u C-c C-e C-t*, visit the generated HTML file instead. Requires the Roxygen and tools packages to be installed.

`ess-roxy-previous-entry` [Command]

C-c C-o p Go to start of the Roxygen entry above point.

`ess-roxy-next-entry` [Command]

C-c C-o n Go to end of the Roxygen entry above point.

`ess-roxy-hide-all` [Command]

C-c C-o C-h Use the hideshow mode to fold away the visibility of all Roxygen entries. Hide-show support must be enabled for this binding to get defined.

ESS also advises the following standard editing functions in order to make Roxygen editing more intuitive:

TAB `ess-R-complete-object-name` Complete Roxygen tag at point. E.g. doing *TAB* when the point is at the end of `@par` completes to `@param`.

M-h `mark-paragraph` If the transient mark mode is active, place mark and point at start end end of the field at point and activate the mark.

TAB `ess-indent-command` If hide-show support is enabled, fold away the visibility of the Roxygen entry at point.

M-q `fill-paragraph` Fill the Roxygen field at point.

C-a `move-beginning-of-line` Move to the point directly to the right of the Roxygen start string.

RET `newline-and-indent` Insert a new line and the Roxygen prefix string.

10.3 Namespaced Evaluation

In non package files evaluation commands, See Chapter 5 [Evaluating code], page 24, send portions of the current buffer environment (`R_GlobalEnv`. When developing packages, ESS sends code to the corresponding package namespace and (for visible objects) into package environment (visible on search path). All objects that are assigned are displayed in the minibuffer alongside the environment in which they are assigned.

Here is a short overview of how namespace and package environments work in R. All objects defined in a package 'foo' are stored in an environment called 'namespace:foo'. Parent environment of 'namespace:foo' is an environment 'imports:foo' which contains copies of all objects from other packages which 'foo' imports. Parent environment of 'imports:foo' is the 'namespace:base'. Parent environment of 'namespace:base' is .GlobalEnv. Thus functions and methods stored in 'namespace:foo' see all the objects in .GlobalEnv unless shadowed by objects in 'imports:foo', 'namespace:base', or 'namespace:foo' itself. There is another environment associated with 'foo' - 'package:foo'. This environment stores *copies* of exported objects from 'namespace:foo' and is placed on the search() path, i.e. if 'foo' is loaded and if you start with .GlobalEnv and iteratively call parent.env() you will get eventually to 'package:foo'. Thus all methods and functions defined in .GlobalEnv can "see" objects in 'package:foo' environment. See also `https://cran.r-project.org/doc/manuals/R-ints.html#Namespaces`.

Ocasionally you want to evaluate into a package from a non-package file, or the other way around, evaluate into `GlobalEnv` from inside a package. In such cases *C-c C-t C-s* is your friend.

`ess-r-set-evaluation-env` *arg* [Command]

C-c C-t C-s Set or unset the current evaluation environment (a package).

11 Other ESS features and tools

ESS has a few extra features, which didn't fit anywhere else.

11.1 ElDoc

In `ElDoc` mode, the echo area displays function's arguments at point. ElDoc is active by default in `ess-mode` and `inferior-ess-mode` buffers. To activate it only in `ess-mode` buffers, place the following into your init file:

```
(setq ess-use-eldoc 'script-only)
```

`ess-use-eldoc` [User Option]
: If '`t`', activate eldoc in ess-mode and inferior-ess-mode buffers. If ''`script-only`' activate in ess-mode buffers only. Set `ess-use-eldoc` to `nil` to stop using `ElDoc` altogether.

`ess-eldoc-show-on-symbol` [User Option]
: This variable controls whether the help is shown only inside function calls. If set to '`t`', `ElDoc` shows help string whenever the point is on a symbol, otherwise (the default), shows only when the point is in a function call, i.e. after ''`(`''.

`ess-eldoc-abbreviation-style` [User Option]
: The variable determines how the doc string should be abbreviated to fit into minibuffer. Posible values are '`nil`', '`mild`', '`normal`', '`strong`' and '`aggressive`'. Please see the documentation of the variable for more details. The default filter is '`normal`'.

Ess-eldoc also honors the value of `eldoc-echo-area-use-multiline-p`, which if set to '`nil`', will cause the truncation of doc string indifferent of the value of `ess-eldoc-abbreviation-style`. This way you can combine different filter levels with the truncation.

11.2 Flymake

The minor mode `flymake-mode` provides on-the-fly syntax checking. ESS provides support for flymake in R-mode in Emacs versions 26 and newer. It is enabled by default, to disable it you may set `ess-use-flymake` to `nil`. In order to use it, you may need to install the `lintr` R package, available from CRAN.

`ess-use-flymake` [User Option]
: When non-nil, use flymake. If `'process`, only use flymake when the buffer has an inferior process running.

`ess-r-flymake-linters` [User Option]
: This variable describes the linters to use. It can either be a string with an R expression to be used as-is or a list of strings where each element is passed to `lintr::with_defaults`. See the help page for `lintr::default_linters` for information on available linters and their defaults.

`ess-r-flymake-lintr-cache` [User Option]
: When `t` (the default), lintr uses a cache. Change to `nil` to disable lintr's caching mechanism.

11.3 Handy commands

`ess-handy-commands` [Command]
Request and execute a command from `ess-handy-commands` list.

`ess-handy-commands` [User Option]
An alist of custom ESS commands available for call by `ess-handy-commands` and `ess-smart-comma` function.

Currently contains:

change-directory `ess-change-directory`

help-index `ess-display-index`

help-object `ess-display-help-on-object`

vignettes `ess-display-vignettes`

objects[ls] `ess-execute-objects`

search `ess-execute-search`

set-width `ess-execute-screen-options`

install.packages `ess-install.packages`

library `ess-library`

setRepos `ess-setRepositories`

sos `ess-sos`

Handy commands: `ess-library`, `ess-install.packages`, etc - ask for item with completion and execute the correspond command. `ess-sos` is a interface to `findFn` function in package `sos`. If package `sos` is not found, ask user for interactive install.

11.4 Syntactic highlighting of buffers

ESS provides Font-Lock (see Section "Using Multiple Typefaces" in *The Gnu Emacs Reference Manual*) patterns for inferior R Mode, S Mode, and R Transcript Mode buffers.

The font-lock patterns are defined by the following variables, which you may modify if desired:

`inferior-R-font-lock-keywords` [User Option]
Font-lock patterns for inferior *R* processes. (There is a corresponding `inferior-S-font-lock-keywords` for *S* processes.) The default value highlights prompts, inputs, assignments, output messages, vector and matrix labels, and literals such as '`NA`' and `TRUE`.

`ess-R-font-lock-keywords` [User Option]
Font-lock patterns for ESS R programming mode. (There is a corresponding `ess-S-font-lock-keywords` for S buffers.) The default value highlights function names, literals, assignments, source functions and reserved words.

11.5 Parenthesis matching

Emacs has facilities for highlighting the parenthesis matching the parenthesis at point. This feature is very useful when trying to examine which parentheses match each other. This highlighting also indicates when parentheses are not matching. You may activate it putting this in your Emacs configuration file:

```
(show-paren-mode)
```

11.6 Using graphics with ESS

One of the main features of R is its ability to generate high-resolution graphics plots. When using R in a windowing environment, no additional configuration is necessary; plots will be shown in a new (non-Emacs) window.

If not using a windowing environment or S, ESS provides a number of features for dealing with such plots.

11.6.1 Using ESS with the `printer()` driver

This is the simplest (and least desirable) method of using graphics within ESS. S's `printer()` device driver produces crude character based plots which can be contained within the ESS process buffer itself. To start using character graphics, issue the S command

```
printer(width=79)
```

(the `width=79` argument prevents Emacs line-wrapping at column 80 on an 80-column terminal. Use a different value for a terminal with a different number of columns.) Plotting commands do not generate graphics immediately, but are stored until the `show()` command is issued, which displays the current figure.

11.6.2 Using ESS with windowing devices

Of course, the ideal way to use graphics with ESS is to use a windowing system. Under X Windows, or X11, this requires that the DISPLAY environment variable be appropriately set.

11.6.3 Java Graphics Device

S+6.1 and newer on Windows contains a java library that supports graphics. Send the commands:

```
library(winjava)
java.graph()
```

to start the graphics driver. This allows you to use ESS for both interaction and graphics within S-PLUS. (Thanks to Tim Hesterberg for this information.)

11.7 Imenu

Imenu is an Emacs tool for providing mode-specific buffer indexes. In some of the ESS editing modes (currently SAS and R), support for Imenu is provided. For example, in R mode buffers, the menubar should display an item called "Imenu-R". Within this menubar you will then be offered bookmarks to particular parts of your source file (such as the starting point of each function definition).

Imenu works by searching your buffer for lines that match what ESS thinks is the beginning of a suitable entry, for example the beginning of a function definition. To examine the regular expression that ESS uses, check the value of `imenu-generic-expression`. This value is set by various ESS variables such as `ess-imenu-S-generic-expression`.

11.8 Toolbar

The R and S editing modes have support for a toolbar. This toolbar provides icons to act as shortcuts for starting new S/R processes, or for evaluating regions of your source buffers. The toolbar should be present if your Emacs can display images. See Appendix A [Customization], page 68, for ways to change the toolbar.

11.9 Xref

Xref is an Emacs interface that supports finding "identifiers," usually function definitions in ESS's view. ESS ships with support for Xref in Emacs versions 25.1 and higher. See Section "Xref" in *The Gnu Emacs Reference Manual* for how to use this feature.

11.10 Rdired

Ess-rdired provides a dired-like buffer for viewing, editing and plotting objects in your current R session. If you are used to using the dired (directory editor) facility in Emacs, this mode gives you similar functionality for R objects.

Start an R session with *M-x R* and then store a few variables, such as:

```
s <- sin(seq(from=0, to=8*pi, length=100))
x <- c(1, 4, 9)
y <- rnorm(20)
z <- TRUE
```

Then use *M-x ess-rdired* to create a buffer listing the objects in your current environment and display it in a new window:

```
             mode length
s       numeric    100
x       numeric      3
y       numeric     20
z       logical      1
```

Type *C-h m* or *?* to get a list of the keybindings for this mode. For example, with your point on the line of a variable, 'p' will plot the object, 'v' will view it, and 'd' will mark the object for deletion ('x' will actually perform the deletion).

11.11 Rutils

Ess-rutils builds up on ess-rdired, providing key bindings for performing basic R functions in the inferior-ESS process buffer, such as loading and managing packages, object manipulation (listing, viewing, and deleting), and alternatives to `help.start()` and `RSiteSearch()` that use the `browse-url` Emacs function. The library can be loaded using *M-x load-file*, but the easiest is to include:

```
(require 'ess-rutils)
```

in your Emacs configuration file. Once R is started with *M-x R*, and if the value of the customizable variable `ess-rutils-keys` is true, several key bindings become available in iESS process buffers:

`ess-rutils-local-pkgs` [Command]
C-c C-. l List all packages in all available libraries.

`ess-rutils-repos-pkgs` [Command]
C-c C-. r List available packages from repositories listed by `getOptions(``repos'')` in the current R session.

`ess-rutils-update-pkgs` *lib repos* [Command]
C-c C-. u Update packages in a particular library *lib* and repository *repos*.

`ess-rutils-apropos` [Command]
C-c C-. a Search for a string using apropos.

`ess-rutils-rm-all` [Command]
C-c C-. m Remove all R objects.

`ess-rutils-objs` [Command]
C-c C-. o Manipulate R objects; wrapper for `ess-rdired`.

`ess-rutils-load-wkspc` [Command]
C-c C-. w Load a workspace file into R.

`ess-rutils-save-wkspc` [Command]
C-c C-. s Save a workspace file.

`ess-change-directory` [Command]
C-c C-. d Change the working directory for the current R session.

`ess-rutils-html-docs` [Command]
C-c C-. H Use `browse-url` to navigate R html documentation.

See the submenu '`Rutils`' under the iESS menu for alternative access to these functions. The function `ess-rutils-rsitesearch` is provided without a particular key binding. This function is useful in any Emacs buffer, so can be bound to a user-defined key:

```
(eval-after-load "ess-rutils"
  '(global-set-key [(control c) (f6)] 'ess-rutils-rsitesearch))
```

Functions for listing objects and packages (`ess-rutils-local-pkgs`, `ess-rutils-repos-pkgs`, and `ess-rutils-objs`) show results in a separate buffer and window, in `ess-rutils-mode`, providing useful key bindings in this mode (type `?` in this buffer for a description).

11.12 Interaction with Org mode

Org-mode (`https://orgmode.org`) now supports reproducible research and literate programming in many languages (including R) – see chapter 14 of the Org manual (`https://orgmode.org/org.html#Working-With-Source-Code`). For ESS users, this offers a document-based work environment within which to embed ESS usage. R code lives in code blocks of an Org document, from which it can be edited in ess-mode, evaluated, extracted ("tangled") to pure code files. The code can also be exported ("woven") with the surrounding text to several formats including HTML and LaTeX. Results of evaluation including figures can be captured in the Org document, and data can be passed from the Org document (e.g. from a table) to the ESS R process. (This section contributed by Dan Davison and Eric Schulte.)

11.13 Support for Sweave in ESS and AUCTeX

ESS provides support for writing and processing Sweave (`https://www.statistik.lmu.de/~leisch/Sweave`), building up on Emacs' ess-noweb-mode for literate programming. When working on an Sweave document, the following key bindings are available:

`ess-swv-weave` *choose* [Command]
: *M-n s* Run Sweave on the current .Rnw file. If *choose* is non-'`nil`', offer a menu of available weavers.

`ess-swv-latex` [Command]
: *M-n l* Run LaTeX after Sweave'ing.

`ess-swv-PS` [Command]
: *M-n p* Generate and display a postscript file after LaTeX'ing.

`ess-swv-PDF` *pdflatex-cmd* [Command]
: *M-n P* Generate and display a PDF file after LaTeX'ing. Optional argument *pdflatex-cmd* is the command to use, which by default, is the command used to generate the PDF file is the first element of `ess-swv-pdflatex-commands`.

`ess-swv-pdflatex-commands` [User Option]
: Commands used by `ess-swv-PDF` to run a version of pdflatex; the first entry is the default command.

Sweave'ing with `ess-swv-weave` starts an inferior-ESS process, if one is not available. Other commands are available from the '`Sweaving, Tangling, ...`' submenu of the Noweb menu.

AUCTeX (`https://www.gnu.org/software/auctex`) users may prefer to set the variable `ess-swv-plug-into-AUCTeX-p` (available from the "ESS Sweave" customization group) to t. Alternatively, the same can be achieved by activating the entry "AUCTeX Interface" from the '`Sweaving, Tangling, ...`' submenu, which toggles this variable on or off. When the interface is activated, new entries for Sweave'ing and LaTeX'ing thereafter are available from AUCTeX's "Command" menu. Sweave'ing can, thus, be done by *C-c C-c Sweave RET* without an inferior-ESS process. Similarly, LaTeX'ing can be done by *C-c C-c LaTeXSweave RET*. In both cases, the process can be monitored with *C-c C-l* (`TeX-recenter-output-buffer`). Open the viewer with *C-c C-v* (`TeX-view`), as usual in AUCTeX.

12 Overview of ESS features for the S family

12.1 ESS[R]–Editing files

ESS[R] mode should be automatically turned on when visiting a file ending with an R or S suffix (*.R, *.S, *.s, etc), which enables the features discussed previously. Alternatively, type *M-x R-mode* to put the current buffer into R mode. However, one will have to start up an inferior process to take advantage of the interactive features.

12.2 iESS[R]–Inferior ESS processes

To start up iESS mode for R, use:

```
M-x R
M-x S+3
M-x S4
M-x S+5
M-x S+6
```

(for R, S-PLUS 3.x, S4, S+5, S+6 or or S+7, respectively. This assumes that you have access to each).

In the case that you wish to pass command line arguments to the starting R process, call it with the universal prefix. To set command line arguments that apply to all future iESS sessions, set the variable `inferior-R-args`.

Note that R has some extremely useful command line arguments. For example, `--vanilla` will ensure R starts up without loading in any init files.

If you have other versions of R or S available on the system, ESS is also able to start those versions. How this exactly works depend on which OS you are using (details below). The general principle, regardless of OS, is that ESS searches the paths listed in the variable `exec-path` for R binaries. If ESS cannot find your R binaries, on Unix you can change the unix environment variable `PATH`, as this variable is used to set `exec-path`.

R on GNU/Linux systems and other Unix-like systems (macOS): If you have `"R-1.8.1"` on your `exec-path`, it can be started using *M-x R-1.8.1*. By default, ESS will find versions of R beginning "R-1", "R-2", "R-3", "R-devel", or "R-patched". If your versions of R are called other names, consider renaming them with a symbolic link or change the variable `ess-r-versions`. To see which functions have been created for starting different versions of R, type *M-x R-* and then hit [Tab]. These other versions of R can also be started from the `"ESS->Start Process->Other"` menu.

R on Windows systems: If you have `"rw1081"` on your `exec-path`, it can be started using *M-x rw1081*. By default, ESS will find versions of R located in directories parallel to the version of R in your `PATH`. If your versions of R are called other names, you will need to change the variable `ess-rterm-versions`. To see which functions have been created for starting different versions of R, type *M-x rw* and then hit [Tab]. These other versions of R can also be started from the `"ESS->Start Process->Other"` menu.

Once ESS has found these extra versions of R, it will then create a new function, called *M-x R-newest*, which will call the newest version of R that it found. (ESS examines the date in the first line of information from `R --version` to determine which is newest.)

S on Unix systems: If you have "Splus7" on your `exec-path`, it can be started using *M-x Splus7*. By default, ESS will find all executables beginning "Splus" on your path. If your versions of S are called other names, consider renaming them with a symbolic link or change the variable `ess-s-versions`. To see which functions have been created for starting different versions of Splus, type *M-x Splus* and then hit [Tab]. These other versions of Splus can also be started from the "ESS->Start Process->Other" menu.

A second mechanism is also available for running other versions of Splus. The variable `ess-s-versions-list` is a list of lists; each sublist should be of the form: (DEFUN-NAME PATH ARGS). DEFUN-NAME is the name of the new emacs function you wish to create to start the new S process; PATH is the full path to the version of S you want to run; ARGS is an optional string of command-line arguments to pass to the S process. Here is an example setting:

```
(setq ess-s-versions-list
      '( ("Splus64" "/usr/local/bin/Splus64")
         ("Splus64-j" "/usr/local/bin/Splus64" "-j")))
```

which will then allow you to do *M-x Splus64-j* to start Splus64 with the corresponding command line arguments.

If you change the value of either `ess-r-versions` or `ess-s-versions-list`, you should put them in your Emacs configuration file before ess-site is loaded, since the new Emacs functions are created when ESS is loaded.

Sqpe (S-Plus running inside an Emacs buffer) on Windows systems: If you have an older version of S-Plus (S-Plus 6.1 for example) on your system, ir can be started inside an Emacs buffer with *M-x splus61*. By default, ESS will find versions of S-Plus located in the installation directories that Insightful uses by default. If your versions of S-Plus are anywhere else, you will need to change the variable `ess-SHOME-versions`. To see which functions have been created for starting different versions of S-Plus, type *M-x spl* and then hit [Tab]. These other versions of S-Plus can also be started from the "ESS->Start Process->Other" menu.

12.3 Philosophies for using ESS[R]

There are two philosophies for using ESS. Most modern best practices prefer the first. ESS is configured for the first, and this is what the manual focuses on.

1: The source code is real. Objects are realizations of the source code. Source for EVERY user modified object is placed in a particular directory or directories, for later editing and retrieval.

2: R objects are real. Source code is a temporary realization of the objects. Dumped buffers should not be saved. _We_strongly_discourage_this_approach_. However, if you insist, add the following lines to your Emacs configuration file:

```
(setq ess-keep-dump-files nil)
(setq ess-delete-dump-files t)
(setq ess-mode-silently-save nil)
```

12.4 Example ESS usage

We present some basic examples for using ESS to interact with R These are just a subset of approaches, many better approaches are possible. Contributions of examples of how you work with ESS are appreciated (especially since it helps us determine priorities on future enhancements)! Comments as to what should be happening are prefixed by "##".

1: ## Data Analysis Example
Load the file you want to work with
C-x C-f myfile.R

Edit as appropriate, then start and switch to an R buffer
C-c C-z

A new buffer *R* will appear. R will have been started
in this buffer. The buffer is in iESS [R] mode.

Return to the script (prefix with C-c if you have pressed other keys)
C-z

Send regions, lines, or the entire file contents to R. For regions,
highlight a region with keystrokes or mouse and then send with:
C-c C-r

Re-edit myfile.R as necessary to correct any difficulties. Add
new commands here. Send them to R by region with C-c C-r, or
one line at a time with C-c C-n.

Save the revised myfile.R with C-x C-s.

Save the entire *R* interaction buffer with C-c C-s. You
will be prompted for a file name. The recommended name is
myfile.Rout. With this suffix, the file will come up in ESS
Transcript mode the next time it is accessed from Emacs.

2: ## Program revision example (source code is real)

Start R in a process buffer (this will be *R*)
M-x R

Load the file you want to work with
C-x C-f myfile.R

edit program, functions, and code in myfile.R, and send revised
functions to R when ready with
C-c C-f

```
## or highlighted regions with
C-c C-r
## or individual lines with
C-c C-n
## or load the entire buffer with
C-c M-l

## save the revised myfile.R when you have finished
C-c C-s
```

13 ESS for SAS

ESS[SAS] was designed for use with SAS. It is descended from emacs macros developed by John Sall for editing SAS programs and `SAS-mode` by Tom Cook. Those editing features and new advanced features are part of ESS[SAS]. The user interface of ESS[SAS] has similarities with ESS[S] and the SAS Display Manager.

13.1 ESS[SAS]–Design philosophy

ESS[SAS] was designed to aid the user in writing and maintaining SAS programs, such as *`foo`*`.sas`. Both interactive and batch submission of SAS programs is supported.

ESS[SAS] was written with two primary goals.

1. The emacs text editor provides a powerful and flexible development environment for programming languages. These features are a boon to all programmers and, with the help of ESS[SAS], to SAS users as well.
2. Although a departure from SAS Display Manager, ESS[SAS] provides similar key definitions to give novice ESS[SAS] users a head start. Also, inconvenient SAS Display Manager features, like remote submission and syntax highlighting, are provided transparently; appealing to advanced ESS[SAS] users.

13.2 ESS[SAS]–Editing files

ESS[SAS] is the mode for editing SAS language files. This mode handles:

- proper indenting, generated by both `TAB` and `RET`.
- color and font choices based on syntax.
- ability to save and submit the file you are working on as a batch SAS process with a single keypress and to continue editing while it is runs in the background.
- capability of killing the batch SAS process through the `*shell*` buffer or allow the SAS process to keep on running after you exit emacs.
- single keypress navigation of `.sas`, `.log` and `.lst` files (`.log` and `.lst` files are refreshed with each keypress).
- ability to send the contents of an entire buffer, a highlighted region, or a single line to an interactive SAS process.
- ability to switch between processes which would be the target of the buffer (for the above).

ESS[SAS] is automatically turned on when editing a file with a `.sas` suffix (or other extension, if specified via `auto-mode-alist`). The function keys can be enabled to use the same function keys that the SAS Display Manager does. The interactive capabilities of ESS require you to start an inferior SAS process with *`M-x SAS`* (See Section 13.6 [iESS(SAS)–Interactive SAS processes], page 61.)

At this writing, the indenting and syntax highlighting are generally correct. Known issues: for multiple line `*` or `%*` comments, only the first line is highlighted; for `.log` files, only the first line of a `NOTE:`, `WARNING:` or `ERROR:` message is highlighted; unmatched single/double quotes in `CARDS` data lines are **NOT** ignored; in an iterative `DO` statement, `TO` and `BY` are not highlighted.

13.3 ESS[SAS]–TAB key

Two options. The TAB key is bound by default to sas-indent-line. This function is used to syntactically indent SAS code so PROC and RUN are in the left margin, other statements are indented sas-indent-width spaces from the margin, continuation lines are indented sas-indent-width spaces in from the beginning column of that statement. This is the type of functionality that emacs provides in most programming language modes. This functionality is activated by placing the following line in your initialization file prior to a require/load:

```
(setq ess-sas-edit-keys-toggle nil)
```

ESS provides an alternate behavior for TAB that makes it behave as it does in SAS Display Manager, i.e. move the cursor to the next stop. The alternate behavior also provides a "TAB" backwards, *C-TAB*, that moves the cursor to the stop to the left and deletes any characters between them. This functionality is obtained by placing the following line in your initialization file prior to a require/load:

```
(setq ess-sas-edit-keys-toggle t)
```

Under the alternate behavior, TAB is bound to *M-x tab-to-tab-stop* and the stops are defined by ess-sas-tab-stop-list.

13.4 ESS[SAS]–Batch SAS processes

Submission of a SAS batch job is dependent on your environment. ess-sas-submit-method is determined by your operating system and your shell. It defaults to 'sh unless you are running Windows or Mac Classic. Under Windows, it will default to 'sh if you are using a UNIX-imitating shell; otherwise 'ms-dos for an MS-DOS shell. On macOS, it will default to 'sh, but under Mac Classic, it defaults to 'apple-script. You will also set this to 'sh if the SAS batch job needs to run on a remote machine rather than your local machine. This works transparently if you are editing the remote file via ange-ftp/EFS or tramp. Note that ess-sas-shell-buffer-remote-init is a Local Variable that defaults to "ssh" which will be used to open the buffer on the remote host and it is assumed that no password is necessary, i.e. you are using ssh-agent/ssh-add or the equivalent (see the discussion about Local Variables below if you need to change the default).

However, if you are editing the file locally and transferring it back and forth with Kermit, you need some additional steps. First, start Kermit locally before remotely logging in. Open a local copy of the file with the ess-kermit-prefix character prepended (the default is "#"). Execute the command *M-x ess-kermit-get* which automatically brings the contents of the remote file into your local copy. If you transfer files with Kermit manually in a *shell* buffer, then note that the Kermit escape sequence is *C-q C-\ c* rather than *C-\ c* which it would be in an ordinary terminal application, i.e. not in an emacs buffer. Lastly, note that the remote Kermit command is specified by ess-kermit-command.

The command used by the SUBMIT function key (F3 or F8) to submit a batch SAS job, whether local or remote, is ess-sas-submit-command which defaults to sas-program. sas-program is "invoke SAS using program file" for Mac Classic and "sas" otherwise. However, you may have to alter ess-sas-submit-command for a particular program, so it is defined as buffer-local. Conveniently, it can be set at the end of the program:

```
endsas;
```

```
Local variables:
ess-sas-submit-command: "sas8"
End:
```

The command line is also made of `ess-sas-submit-pre-command`, `ess-sas-submit-post-command` and `ess-sas-submit-command-options` (the last of which is also buffer-local). Here are some examples for your `~/.emacs` or `~/.xemacs/init.el` file (you may also use *M-x customize-variable*):

```
;'sh default
(setq ess-sas-submit-pre-command "nohup")
;'sh default
(setq ess-sas-submit-post-command "-rsasuser &")
;'sh example
(setq-default ess-sas-submit-command "/usr/local/sas/sas")
;'ms-dos default
(setq ess-sas-submit-pre-command "start")
;'ms-dos default
(setq ess-sas-submit-post-command "-rsasuser -icon")
;Windows example
(setq-default ess-sas-submit-command "c:/progra~1/sas/sas.exe")
;Windows example
(setq-default ess-sas-submit-command "c:\\progra~1\\sas\\sas.exe")
```

There is a built-in delay before a batch SAS job is submitted when using a UNIX-imitating shell under Windows. This is necessary in many cases since the shell might not be ready to receive a command. This delay is currently set high enough so as not to be a problem. But, there may be cases when it needs to be set higher, or could be set much lower to speed things up. You can over-ride the default in your `~/.emacs` or `~/.xemacs/init.el` file by:

```
(setq ess-sleep-for 0.2)
```

For example, `(setq ess-sas-global-unix-keys t)` keys shown, `(setq ess-sas-global-pc-keys t)` in parentheses; ESS[SAS] function keys are presented in the next section. Open the file you want to work with *C-x C-f foo.sas*. *foo*`.sas` will be in ESS[SAS] mode. Edit as appropriate, then save and submit the batch SAS job.

```
F3 (F8)
```

The job runs in the `*shell*` buffer while you continue to edit *foo*`.sas`. If `ess-sas-submit-method` is `'sh`, then the message buffer will display the shell notification when the job is complete. The `'sh` setting also allows you to terminate the SAS batch job before it is finished.

```
F8 (F3)
```

Terminating a SAS batch in the `*shell*` buffer.

```
kill PID
```

You may want to visit the `.log` (whether the job is still running or it is finished) and check for error messages. The `.log` will be refreshed and you will be placed in it's buffer. You will be taken to the first error message, if any.

```
F5 (F6)
```

Goto the next error message, if any.

```
F5 (F6)
```

Now, '`refresh`' the `.lst` and go to it's buffer.

```
F6 (F7)
```

If you wish to make changes, go to the `.sas` file with.

```
F4 (F5)
```

Make your editing changes and submit again.

```
F3 (F8)
```

13.5 ESS[SAS]–Function keys for batch processing

The setup of function keys for SAS batch processing is unavoidably complex, but the usage of function keys is simple. There are five distinct options:

Option 1 (default). Function keys in ESS[SAS] are not bound to elisp commands. This is in accordance with the GNU Elisp Coding Standards (GECS) which do not allow function keys to be bound so that they are available to the user.

Options 2-5. Since GECS does not allow function keys to be bound by modes, these keys are often unused. So, ESS[SAS] provides users with the option of binding elisp commands to these keys. Users who are familiar with SAS will, most likely, want to duplicate the function key capabilities of the SAS Display Manager. There are four options (noted in parentheses).

a. SAS Display Manager has different function key definitions for UNIX (2, 4) and Windows (3, 5); ESS[SAS] can use either.
b. The ESS[SAS] function key definitions can be active in all buffers (global: 4, 5) or limited (local: 2, 3) only to buffers with files that are associated with ESS[SAS] as specified in your `auto-mode-alist`.

The distinction between local and global is subtle. If you want the ESS[SAS] definitions to work when you are in the `*shell*` buffer or when editing files other than the file extensions that ESS[SAS] recognizes, you will most likely want to use the global definitions. If you want your function keys to understand SAS batch commands when you are editing SAS files, and to behave normally when editing other files, then you will choose the local definitions. The option can be chosen by the person installing ESS for a site or by an individual.

a. For a site installation or an individual, place **ONLY ONE** of the following lines in your initialization file prior to a `require/load`. ESS[SAS] function keys are available in ESS[SAS] if you choose either 2 or 3 and in all modes if you choose 4 or 5:

```
;;2; (setq ess-sas-local-unix-keys t)
;;3; (setq ess-sas-local-pc-keys t)
;;4; (setq ess-sas-global-unix-keys t)
;;5; (setq ess-sas-global-pc-keys t)
```

   The names `-unix-` and `-pc-` have nothing to do with the operating system that you are running. Rather, they mimic the definitions that the SAS Display Manager uses by default on those platforms.

b. If your site installation has configured the keys contrary to your liking, then you must call the appropriate function.

```
(load "ess-site") ;; local-unix-keys
```

```
(ess-sas-global-pc-keys)
```

Finally, we get to what the function keys actually do. You may recognize some of the nicknames as SAS Display Manager commands (they are in all capitals).

UNIX	PC	Nickname
F2	F2	'refresh' revert the current buffer with the file of the same name if the file is newer than the buffer
F3	F8	SUBMIT save the current .sas file (which is either the .sas file in the current buffer or the .sas file associated with the .lst or .log file in the current buffer) and submit the file as a batch SAS job
F4	F5	PROGRAM switch buffer to .sas file
F5	F6	LOG switch buffer to .log file, 'refresh' and goto next error message, if any
F6	F7	OUTPUT switch buffer to .lst file and 'refresh'
F7	F4	'filetype-1' switch buffer to 'filetype-1' (defaults to .txt) file and 'refresh'
F8	F3	'shell' switch buffer to *shell*
F9	F9	VIEWTABLE open an interactive PROC FSEDIT session on the SAS dataset near point
F10	F10	toggle-log toggle ESS[SAS] for .log files; useful for certain debugging situations
F11	F11	'filetype-2' switch buffer to 'filetype-2' (defaults to .dat) file and 'refresh'
F12	F12	viewgraph open a GSASFILE near point for viewing either in emacs or with an external viewer
C-F1	*C-F1*	rtf-portrait create an MS RTF portrait file from the current buffer with a file extension of .rtf
C-F2	*C-F2*	rtf-landscape create an MS RTF landscape file from the current buffer with a file extension of .rtf
C-F3	*C-F8*	submit-region write region to ess-temp.sas and submit
C-F5	*C-F6*	append-to-log append ess-temp.log to the current .log file
C-F6	*C-F7*	append-to-output append ess-temp.lst to the current .lst file
C-F9	*C-F9*	INSIGHT

		open an interactive `PROC INSIGHT` session on the SAS dataset near point
C-F10	*C-F10*	toggle-listing
		toggle ESS[SAS] for `.lst` files; useful for toggling read-only

`SUBMIT`, `PROGRAM`, `LOG` and `OUTPUT` need no further explanation since they mimic the SAS Display Manager commands and related function key definitions. However, six other keys have been provided for convenience and are described below.

'`shell`' switches you to the `*shell*` buffer where you can interact with your operating system. This is especially helpful if you would like to kill a SAS batch job. You can specify a different buffer name to associate with a SAS batch job (besides `*shell*`) with the buffer-local variable `ess-sas-shell-buffer`. This allows you to have multiple buffers running SAS batch jobs on multiple local/remote computers that may rely on different methods specified by the buffer-local variable `ess-sas-submit-method`.

`F2` performs the '`refresh`' operation on the current buffer. '`refresh`' compares the buffer's last modified date/time with the file's last modified date/time and replaces the buffer with the file if the file is newer. This is the same operation that is automatically performed when `LOG`, `OUTPUT`, '`filetype-1`' or `F11` are pressed.

'`filetype-1`' switches you to a file with the same file name as your `.sas` file, but with a different extension (`.txt` by default) and performs '`refresh`'. You can over-ride the default extension; for example in your `~/.emacs` or `~/.xemacs/init.el` file:

```
(setq ess-sas-suffix-1 "csv") ; for example
```

`F9` will prompt you for the name of a permanent SAS dataset near point to be opened for viewing by `PROC FSEDIT`. You can control the SAS batch command-line with `ess-sas-data-view-submit-options`. For controlling the SAS batch commands, you have the global variables `ess-sas-data-view-libname` and `ess-sas-data-view-fsview-command` as well as the buffer-local variable `ess-sas-data-view-fsview-statement`. If you have your SAS `LIBNAME` defined in `~/autoexec.sas`, then the defaults for these variables should be sufficient.

Similarly, *C-F9* will prompt you for the name of a permanent SAS dataset near point to be opened for viewing by `PROC INSIGHT`. You can control the SAS batch command-line with `ess-sas-data-view-submit-options`. For controlling the SAS batch commands, you have the global variables `ess-sas-data-view-libname` and `ess-sas-data-view-insight-command` as well as the buffer-local variable `ess-sas-data-view-insight-statement`.

`F10` toggles ESS[SAS] mode for `.log` files which is off by default (technically, it is `SAS-log-mode`, but it looks the same). The syntax highlighting can be helpful in certain debugging situations, but large `.log` files may take a long time to highlight.

`F11` is the same as '`filetype-1`' except it is `.dat` by default.

`F12` will prompt you for the name of a `GSASFILE` near the point in `.log` to be opened for viewing either with emacs or with an external viewer. Depending on your version of emacs and the operating system you are using, emacs may support `.gif` and `.jpg` files internally. You may need to change the following variables for your own situation. `ess-sas-graph-view-suffix-regexp` is a regular expression of supported file types defined via file name extensions. `ess-sas-graph-view-viewer-default` is the default external viewer for your platform. `ess-sas-graph-view-viewer-alist` is an alist of exceptions to the default; i.e. file types and their associated viewers which will be used rather than the default viewer.

```
(setq ess-sas-graph-view-suffix-regexp (concat "[.]\\([eE]?[pP][sS]\\|"
"[pP][dD][fF]\\|[gG][iI][fF]\\|[jJ][pP][eE]?[gG]\\|"
"[tT][iI][fF][fF]?\\)")) ;; default
(setq ess-sas-graph-view-viewer-default "kodakimg") ;; Windows default
(setq ess-sas-graph-view-viewer-default "sdtimage") ;; Solaris default
(setq ess-sas-graph-view-viewer-alist
  '(("[eE]?[pP][sS]" . "gv") ("[pP][dD][fF]" . "gv")) ;; default w/ gv
```

C-F2 produces US landscape by default, however, it can produce A4 landscape (first line for "global" key mapping, second for "local"):

```
(global-set-key [(control f2)] 'ess-sas-rtf-a4-landscape)
(define-key sas-mode-local-map [(control f2)] 'ess-sas-rtf-a4-landscape)
```

13.6 iESS[SAS]–Interactive SAS processes

Inferior ESS (iESS) is the method for interfacing with interactive statistical processes (programs). iESS[SAS] is what is needed for interactive SAS programming. iESS[SAS] works best with the SAS command-line option settings `"-stdio -linesize 80 -noovp -nosyntaxcheck"` (the default of `inferior-SAS-args`).

`-stdio`
: required to make the redirection of stdio work

`-linesize 80`
: keeps output lines from folding on standard terminals

`-noovp`
: prevents error messages from printing 3 times

`-nosyntaxcheck`
: permits recovery after syntax errors

To start up iESS[SAS] mode, use:

M-x SAS

The `*SAS:1.log*` buffer in `ESStr` mode corresponds to the file *foo*`.log` in SAS batch usage and to the '`SAS: LOG`' window in the SAS Display Manager. All commands submitted to SAS, informative messages, warnings, and errors appear here.

The `*SAS:1.lst*` buffer in `ESSlst` mode corresponds to the file *foo*`.lst` in SAS batch usage and to the '`SAS: OUTPUT`' window in the SAS Display Manager. All printed output appears in this window.

The `*SAS:1*` buffer exists solely as a communications buffer. The user should never use this buffer directly. Files are edited in the *foo*`.sas` buffer. The *C-c C-r* key in ESS[SAS] is the functional equivalent of bringing a file into the '`SAS: PROGRAM EDITOR`' window followed by `SUBMIT`.

For example, open the file you want to work with.

C-x C-f foo.sas

foo`.sas` will be in ESS[SAS] mode. Edit as appropriate, and then start up SAS with the cursor in the *foo*`.sas` buffer.

M-x SAS

Four buffers will appear on screen:

Buffer	Mode	Description
foo.sas	ESS[SAS]	your source file
SAS:1	iESS[SAS:1]	iESS communication buffer
SAS:1.log	Shell ESStr []	SAS log information
SAS:1.lst	Shell ESSlst []	SAS listing information

If you would prefer each of the four buffers to appear in its own individual frame, you can arrange for that. Place the cursor in the buffer displaying *foo*.sas. Enter the sequence *C-c C-w*. The cursor will normally be in buffer *foo*.sas. If not, put it there and *C-x b foo.sas*.

Send regions, lines, or the entire file contents to SAS (regions are most useful: a highlighted region will normally begin with the keywords DATA or PROC and end with RUN;), *C-c C-r*.

Information appears in the log buffer, analysis results in the listing buffer. In case of errors, make the corrections in the *foo*.sas buffer and resubmit with another *C-c C-r*.

At the end of the session you may save the log and listing buffers with the usual *C-x C-s* commands. You will be prompted for a file name. Typically, the names *foo*.log and *foo*.lst will be used. You will almost certainly want to edit the saved files before including them in a report. The files are read-only by default. You can make them writable by the emacs command *C-x C-q*.

At the end of the session, the input file *foo*.sas will typically have been revised. You can save it. It can be used later as the beginning of another iESS[SAS] session. It can also be used as a batch input file to SAS.

The *SAS:1* buffer is strictly for ESS use. The user should never need to read it or write to it. Refer to the .lst and .log buffers for monitoring output!

Troubleshooting: See Section 13.7 [iESS(SAS)–Common problems], page 62.

13.7 iESS[SAS]–Common problems

1. iESS[SAS] does not work on Windows. In order to run SAS inside an emacs buffer, it is necessary to start SAS with the -stdio option. SAS does not support the -stdio option on Windows.
2. If *M-x SAS* gives errors upon startup, check the following:
   - you are running Windows: see 1.
   - ess-sas-sh-command (from the ESS etc directory) needs to be executable; too check, type *M-x dired*; if not, fix it as follows, type *M-:*, then at the minibuffer prompt 'Eval:', type (set-file-modes "ess-sas-sh-command" 493).
   - sas isn't in your executable path; to verify, type *M-:* and at the minibuffer prompt 'Eval:', type (executable-find "sas")
3. *M-x SAS* starts SAS Display Manager. Probably, the command sas on your system calls a shell script. In that case you will need to locate the real sas executable and link to it. You can execute the UNIX command:

   ```
   find / -name sas -print
   ```

   Now place a soft link to the real sas executable in your ~/bin directory, with for example

   ```
   cd ~/bin
   ```

```
ln -s /usr/local/sas9/sas sas
```

Check your `PATH` environment variable to confirm that `~/bin` appears before the directory in which the `sas` shell script appears.

13.8 ESS[SAS]–Graphics

Output from a SAS/GRAPH `PROC` can be displayed in a SAS/GRAPH window for SAS batch on Windows or for both SAS batch and interactive with XWindows on UNIX. If you need to create graphics files and view them with `F12`, then include the following (either in *`foo`*`.sas` or in `~/autoexec.sas`):

```
filename gsasfile 'graphics.ps';
goptions device=ps gsfname=gsasfile gsfmode=append;
```

`PROC PLOT` graphs can be viewed in the listing buffer. You may wish to control the vertical spacing to allow the entire plot to be visible on screen, for example:

```
proc plot;
    plot a*b / vpos=25;
run;
```

13.9 ESS[SAS]–Windows

- iESS[SAS] does not work on Windows. See Section 13.7 [iESS(SAS)–Common problems], page 62.
- ESS[SAS] mode for editing SAS language files works very well. See Section 13.2 [ESS(SAS)–Editing files], page 55.
- There are two execution options for SAS on Windows. You can use batch. See Section 13.4 [ESS(SAS)–Batch SAS processes], page 56.

  Or you can mark regions with the mouse and submit the code with 'submit-region' or paste them into SAS Display Manager.

14 ESS for BUGS

ESS[BUGS] provides 5 features. First, BUGS syntax is described to allow for proper fontification of statements, distributions, functions, commands and comments in BUGS model files, command files and log files. Second, ESS creates templates for the command file from the model file so that a BUGS batch process can be defined by a single file. Third, ESS provides a BUGS batch script that allows ESS to set BUGS batch parameters. Fourth, key sequences are defined to create a command file and submit a BUGS batch process. Lastly, interactive submission of BUGS commands is also supported.

14.1 ESS[BUGS]–Model files

Model files with the `.bug` extension are edited in ESS[BUGS] mode. Three keys are bound for your use in ESS[BUGS], *F2*, *C-c C-c* and `=`. *F2* performs the same action as it does in ESS[SAS], See Section 13.5 [ESS(SAS)–Function keys for batch processing], page 58. *C-c C-c* performs the function `ess-bugs-next-action` which you will use a lot. Pressing it in an empty buffer for a model file will produce a template for you. `=` inserts the set operator, `<-`.

14.2 ESS[BUGS]–Command files

Files ending in `.bmd` are used for BUGS command files. When you have finished editing your model file and press *C-c C-c*, a command file is created if one does not already exist. When you are finished editing your command file, pressing *C-c C-c* again will submit your command file as a batch job.

14.3 ESS[BUGS]–Log files

The `.bog` extension is used for BUGS log files. The command line generated by ESS creates the `.bog` transcript file.

15 ESS for JAGS

ESS[JAGS] provides 4 features. First, JAGS syntax is described to allow for proper fontification of statements, distributions, functions, commands and comments in JAGS model files, command files and log files. Second, ESS creates templates for the command file from the model file so that a JAGS batch process can be defined by a single file. Third, ESS provides a JAGS batch script that allows ESS to set JAGS batch parameters. Fourth, key sequences are defined to create a command file and submit a JAGS batch process.

15.1 ESS[JAGS]–Model files

Files with the `.jag` extension are edited in ESS[JAGS] mode. Three keys are bound for your use in ESS[JAGS], *F2*, *C-c C-c* and `=`. *F2* performs the same action as it does in ESS[SAS], See Section 13.5 [ESS(SAS)–Function keys for batch processing], page 58. *C-c C-c* performs the function `ess-bugs-next-action` which you will use a lot. Pressing it in an empty buffer for a model file will produce a template for you. `=` inserts the set operator, `<-`.

The first press of *C-c C-c* will set up a template, including some necessary file-local variables in an empty buffer. These variables are `ess-jags-chains`, `ess-jags-monitor`, `ess-jags-thin`, `ess-jags-burnin` and `ess-jags-update`; they appear in the `Local Variables` section. When you are finished editing your model file, pressing *C-c C-c* will perform the necessary actions to build your command file for you.

The `ess-jags-chains` variable is the number of chains that you want to initialize and sample from; defaults to 1. The `ess-jags-monitor` variable is a list of variables that you want monitored: encase each variable in double quotes. When you press *C-c C-c*, the appropriate statements are created in the command file to monitor the list of variables. By default, no variables are explicitly monitored which means JAGS will implicitly monitor all "default" variables. The `ess-jags-thin` variable is the thinning parameter. By default, the thinning parameter is set to 1, i.e. no thinning. The `ess-jags-burnin` variable is the number of initial samples to discard. By default, the burnin parameter is set to 10000. The `ess-jags-update` variable is the number of post-burnin samples to keep. By default, the update parameter is set to 10000. Both `ess-jags-burnin` and `ess-jags-update` are multiplied by `ess-jags-thin` since JAGS does not do it automatically.

15.2 ESS[JAGS]–Command files

Files ending in `.jmd` are for JAGS command files. For your `.jmd` file, there is only one variable, `ess-jags-command`, in the `Local Variables` section. When you have finished editing your model file and press *C-c C-c*, a command file is created if one does not already exist. When you are finished editing your command file, pressing *C-c C-c* again will submit your command file as a batch job. The `ess-jags-command` variable allows you to specify a different JAGS program to use to run your model; defaults to "jags".

15.3 ESS[JAGS]–Log files

The `.jog` extension is used for JAGS log files. You may find *F2* useful to refresh the `.jog` if the batch process over-writes or appends it.

16 Bugs and Bug Reporting, Mailing Lists

16.1 Bugs

- Commands like `ess-display-help-on-object` and list completion cannot be used while the user is entering a multi-line command. The only real fix in this situation is to use another ESS process.
- The `ess-eval-` commands can leave point in the ESS process buffer in the wrong place when point is at the same position as the last process output. This proves difficult to fix, in general, as we need to consider all *windows* with `window-point` at the right place.
- It's possible to clear the modification flag (say, by saving the buffer) with the edit buffer not having been loaded into S.
- Backup files can sometimes be left behind, even when `ess-keep-dump-files` is `nil`.
- Passing an incomplete S expression to `ess-execute` causes ESS to hang.
- The function-based commands don't always work as expected on functions whose body is not a parenthesized or compound expression, and don't even recognize anonymous functions (i.e. functions not assigned to any variable).
- Multi-line commands could be handled better by the command history mechanism.
- Changes to the continutation prompt in R (e.g. `options(continue = " ")`) are not automatically detected by ESS. Hence, for now, the best thing is not to change the continuation prompt. If you do change the continuation prompt, you will need to change accordingly the value of `inferior-ess-secondary-prompt` in `R-customize-alist`.

16.2 Reporting Bugs

Please send bug reports, suggestions etc. to `ESS-bugs@r-project.org`, or post them on our github issue tracker (`https://github.com/emacs-ess/ESS/issues`)

The easiest way to do this is within Emacs by typing

M-x ess-submit-bug-report

This also gives the maintainers valuable information about your installation which may help us to identify or even fix the bug.

If Emacs reports an error, backtraces can help us debug the problem. Type "M-x set-variable RET debug-on-error RET t RET". Then run the command that causes the error and you should see a *Backtrace* buffer containing debug information; send us that buffer.

Note that comments, suggestions, words of praise and large cash donations are also more than welcome.

16.3 Mailing Lists

There is a mailing list for discussions and announcements relating to ESS. Join the list by sending an e-mail with "subscribe ess-help" (or "help") in the body to `ess-help-request@r-project.org`; contributions to the list may be mailed to `ess-help@r-project.org`. Rest assured, this is a fairly low-volume mailing list.

The purposes of the mailing list include

- helping users of ESS to get along with it.
- discussing aspects of using ESS on Emacs and XEmacs.
- suggestions for improvements.
- announcements of new releases of ESS.
- posting small patches to ESS.

16.4 Help with Emacs

Emacs is a complex editor with many abilities that we do not have space to describe here. If you have problems with other features of Emacs (e.g. printing), there are several sources to consult, including the Emacs FAQs (try *M-x view-emacs-faq*) and EmacsWiki (`https://www.emacswiki.org`). Please consult them before asking on the mailing list about issues that are not specific to ESS.

Appendix A Customizing ESS

ESS can be easily customized to your taste simply by including the appropriate lines in your Emacs configuration file. There are numerous variables which affect the behavior of ESS in certain situations which can be modified to your liking. Keybindings may be set or changed to your preferences, and for per-buffer customizations hooks are also available.

Most of these variables can be viewed and set using the Custom facility within Emacs. Type *M-x customize-group RET ess RET* to see all the ESS variables that can be customized. Variables are grouped by subject to make it easy to find related variables.

Indices

Key index

Function and program index

B

C

D

E

F

G

I

M

N

O

P

Q

R

S

Variable index

A

C

E

I

R

Concept Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

W

X