ESS — Emacs Speaks Statistics

ESS version 5.2.12

The ESS Developers (A.J. Rossini, R.M. Heiberger, K. Hornik, M. Maechler, R.A. Sparapani and S.J. Eglen)
Current Documentation by The ESS Developers

Original Documentation by David M. Smith

1 Introduction to ESS

The S family (S, Splus and R) and SAS statistical analysis packages provide sophisticated statistical and graphical routines for manipulating data. **E**macs **S**peaks **S**tatistics (ESS) is based on the merger of two pre-cursors, S-mode and SAS-mode, which provided support for the S family and SAS respectively. Later on, Stata-mode was also incorporated.

ESS provides a common, generic, and useful interface, through emacs, to many statistical packages. It currently supports the S family, SAS, BUGS, Stata and XLisp-Stat with the level of support roughly in that order.

A bit of notation before we begin. *emacs* refers to both *GNU Emacs* by the Free Software Foundation, as well as *XEmacs* by the XEmacs Project. The emacs major mode `ESS[language]`, where `language` can take values such as `S`, `SAS`, or `XLS`. The inferior process interface (the connection between emacs and the running process) referred to as inferior ESS (`iESS`), is denoted in the modeline by `ESS[dialect]`, where `dialect` can take values such as `S3`, `S4`, `S+3`, `S+4`, `S+5`, `S+6`, `S+7`, `R`, `XLS`, `VST`, `SAS`.

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

For exclusively interactive users of S, 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 S objects and "hot keys" are provided for common S 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 S interactively under ESS.

For those that use S in the typical edit–test–revise cycle when programming S functions, ESS provides for editing of S functions in Emacs edit buffers. Unlike the typical use of S 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 S, and a mechanism to jump directly to the error is provided. ESS also provides for maintaining text versions of your S 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.

ESS provides several features which make it easier to interact with the ESS 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. The '`-e`' flag for S-plus provides something similar to this, but here you have the full range of Emacs commands rather than a limited subset. However, other packages such as XLisp-Stat and S3 do not necessarily have features like this built-in. See Section 4.1 [Command-line editing], page 21.
- **Searchable command history** for recalling previously-submitted commands. This provides all the features of the '`Splus -e`' history mechanism, plus added features such as history searching. See Section 4.5 [Command History], page 25.
- **Command-line completion** of both object and file names for quick entry. This is similar to `tcsh`'s facility for filenames; here it also applies to object names and list components. See Section 4.2 [Completion], page 21.
- **Hot-keys** for quick entry of commonly-used commands in 'S' such as `objects()` and `search()`. See Section 4.7 [Hot keys], page 27.
- **Transcript recording** for a complete record of all the actions in an S session. See Section 4.4 [Transcript], page 23.
- **Interface to the help system**, with a specialized mode for viewing S help files. See Chapter 9 [Help], page 46.

If you commonly create or modify S functions, you will have found the standard facilities for this (the '`fix()`' function, for example) severely limiting. Using S's standard features, one can only edit one function at a time, and you can't continue to use S while editing. ESS corrects these problems by introducing the following features:

- **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 37.
- **A specialized editing mode** for S code, which provides syntactic indentation and highlighting. See Section 7.5 [Indenting], page 39.
- **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 38.
- **Source code revision maintenance**, which allows you to keep historic versions of S source files. See Section 7.7 [Source Files], page 41.
- **Facilities for evaluating S code** such as portions of source files, or line-by-line evaluation of files (useful for debugging). See Section 7.4 [Evaluating code], page 38.

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

- **Evaluation of previously entered commands**, stripping away unnecessary prompts. See Section 4.4.3 [Transcript resubmit], page 24.

1.2 New features in ESS

Changes/New Features in 5.2.12:

- ESS[SAS]: *`M-;`* fixed, but the XEmacs function `comment-dwim` may be broken, if so, use *`M-x comment-region`* and *`M-x uncomment-region`* instead; only valid PROCs are

fontified which is very helpful finding syntax errors (currently supported: BASE, ETS, FSP, GRAPH, IML, INSIGHT and STAT); the "feature" where *F*-keys take you to an empty buffer when the requested destination is a file that does not exist has been fixed, now the request results in a no-op.

Further, sas-mode now also works in simple terminals.

- Rterm/Cygwin combination works under Microsoft Windows.
- ESS[R]: internal calls use baseenv() instead of NULL and define 'baseenv' where needed.
- New experimental support for installing ESS. See the file '`lisp/ess-install.el`'.

Changes/New Features in 5.2.11:

- ESS Info entry and '`dir`' handled more effectively for GNU Emacs users
- ESS[SAS]: temporary files created for batch submission of a region are now named based on the current file; see `ess-sas-file-root` for details; all `lag` and `dif` functions now fontified correctly
- iESS[SAS]: fixed a few nagging bugs, however, still does not appear to work at this time; please let us know if you have any ideas.
- ESS[S]: Support for running other versions of Splus has been added for unix. Two new variables, `ess-s-versions` and `ess-s-versions-list`, are used to tell ESS what other versions of Splus you would like to run.

Changes/New Features in 5.2.10:

- ESS[R]: ess-r-versions can no longer be customized (since the customization was not taking effect unless customizations were loaded before ESS). Its value has been changed so that it will also find R executables beginning "R-devel" and "R-patched". If you wish to change this variable, it must be set in your '`.emacs`' before ESS is loaded.
- Installation with GNU Make enhanced: unix and unix-like operating systems will now be able to install ESS for all users in either a GNU Emacs site-lisp or an XEmacs package configuration by editing '`lisp/ess-site.el`' and '`Makeconf`' accordingly, then issuing `make install`
- ESS[S]: Filename completion (inside strings) now also works in XEmacs for R and S-plus.

Changes/New Features in 5.2.9:

- ESS[R] for Windows: the \ directory character bug with respect to ess-load-file has been eradicated.
- iESS[SAS]: *C-c C-r* and *C-c C-b* once again work as intended and documented.
- ESS[S]: M-x ess-fix-EQ-assign is a bit more agressive.
- ESS[S]: Imenu now also shows setAs(), etc.
- ESS[R]: R function pattern enhanced with underlying code such that *M-C-a* (`ess-beginning-of-function`) etc now work for many more cases, including S4 method definitions.
- iESS[R]: myOwnhelp(1) no longer wrongly triggers help(1).
- ESS[R]: Improved detection of bogus help buffers: valid help buffers containing with the string "no documentation"(e.g. contour) were being treated as bogus.

- ESS[R]: In R help buffers, if `options("help.try.all.packages" = TRUE)` then `?rlm` will list which packages rlm is defined in. This help buffer is not bogus, but instead is now relabelled "*help[R](rlm in packages)*".
- ESS[STA]: add "//" as comment starting character to syntax-table.

Changes/New Features in 5.2.8:

- iESS: [Tab] completes **file** names "inside string" as in earlier (<= 5.2.3) ESS versions.

Changes/New Features in 5.2.7:

- If you use Custom to change the variable ess-toolbar-items, the new toolbar is used in all subsequent ESS buffers.
- ESS[SAS]: new feature: if ess-sas-log-max >0 and your .log grows to more than ess-sas-log-max bytes, just the first ess-sas-log-max bytes are refreshed; this is helpful when your .sas program generates lots of error messages and gets too big for emacs to display
- ESS[R/S]: *M-;* in R/S editing modes will now indent with either one or two hashes depending on context.
- ESS[R]: David Whiting's Sweave extensions (to 'noweb') are now available (from ess-swv.el loaded by default).

Changes/New Features in 5.2.6:

- Removed non-ASCII characters in a few files.
- ESS[R]: now works better when UTF-8 locale is active; in particular, you get correct directional quotes in R's startup message for R-devel (unstable development version of R 2.1.0) when using environment variables LANGUAGE=en@quot LC_ALL=en_US.UTF-8
- ESS[SAS]: toggling of .log mode improved (*F10*); toggling of .lst mode now also available (*C-F10*); killing all buffers associated with .sas program no longer bound to *C-F10* since its a bit overzealous.
- S-Plus 7 for Windows is now recognized.
- ESS[S] (incl. R): in auto-fill mode, strings are not wrapped anymore.
- ESS[S] (incl. R): font-lock now correctly differs between R and S, e.g., for `"_"`; both now fontify warning(.) and S does terminate() additionally.
- Support for 'bell' aka 'beep' aka 'ding' aka 'alarm' in all inferior modes: When \a is output "to the the console" at the beginning of a line, the bell is rung.

Changes/New Features in 5.2.5:

- ESS[R]: *C-c C-q* or '`Quit S`' from the menu now should work (again and less klunkily) and do not append '`-exited`' to the buffer name. Further, the behavior of `(ess-cleanup)`, called from ess-quit, now depends on the new customizable variable `ess-S-quit-kill-buffers-p` which defaults to `nil`. Consequently, the question *"Delete all buffers associated with ..?"* will not be asked anymore by default.
- ESS[SAS] – ess-ebcdic-to-ascii-search-and-replace will now work with the `recode` application as well which is available on many platforms
- ESS[S] (incl. R): Name completion for slots of S4 objects now works!

Changes/New Features in 5.2.4:

- The documentation now includes an overview of how to use the emacs TAGS facility for S functions. (The distribution also used to contain a directory '`etc/other/Tags`' where a ~1990 version of '`etags.c`' was distributed; this is no longer relevant and so has been deleted.)
- ESS[SAS] – When you are working with EBCDIC files on an ASCII platform, .log NOTEs may display as gibberish since the EBCDIC characters are not converted to ASCII prior to their display. So, the function ess-ebcdic-to-ascii-search-and-replace is provided for convenience and is bound to *C-F11*. This function requires the `dd` command (only available on unix or unix-like platforms).
- ESS: Completion of object names is now always done dynamically rather than allowing the option of using a pre-computed database (by `ess-create-object-name-db`) since modern computers seem fast enough for dynamic completion. (We expect few users, if any, have been using the pre-computed database method.)
- ESS: object completion in iESS buffers running on Windows was very slow (for GNU Emacs, but not XEmacs) and has now been fixed. Further, it was more or less broken for all versions of S-plus 6.x, and has been fixed to work everywhere but with the Windows' GUI of S-plus. The list of objects now shows unique names also when an object appears more than once in the search path.
- ESS[R]: Completion of object names now also includes those starting with ".".

Changes/New Features in 5.2.3:

- ESS: When new inferior ESS processes are created, by default they will replace the current buffer (this restores behavior from pre 5.2.0). If you wish new ESS processes to start in another window of the current frame, set inferior-ess-same-window to nil.
- New variables inferior-Splus-args and inferior-R-args provide a way to pass command line arguments to starting S and R processes.

Changes/New Features in 5.2.2:

- bug-fixes for 5.2.1 (require 'executable), html docs, etc.
- ess-lisp-directory/../doc/info added to Info-directory-list if ess-info not found by info
- ESS[R]: If you have other versions of R on your exec-path, such as "R-1.8.1" with Unix or "rw1081" with Windows, ESS will find them and create appropriate functions, such as *M-x R-1.8.1* or *M-x rw1081*, for calling them. By default only Unix programs beginning "R-1" and "R-2" and Windows programs parallel to the version of R in your exec-path will be found, but see ess-r-versions and ess-rterm-versions for ways to find other versions of R.
- ESS[R]: Other versions of R, such as "R-1.8.1" on Unix and "rw1081" on Windows, are added to the "ESS / Start Process / Other" menu.
- ESS[S]: If you have other versions of S-Plus on your Windows computer, such as S-Plus 6.1 or S-Plus 4.5, ESS will find them and create appropriate functions, such as *M-x splus61*, for calling the console version (Sqpe) inside an emacs buffer. By default only programs installed in the default location will be found, but see ess-SHOME-versions for ways to find other versions of S-Plus.
- ESS[S]: Other versions of Sqpe on Windows, such as "splus61", are added to the "ESS / Start Process / Other" menu.

- ESS[R]: (bug fix) ess-quit (bound to *C-c C-q*) should now quit the inferior R process, when issued from either the inferior buffer, or from a .R buffer.

Changes/New Features in 5.2.1:

- ESS[S] (R and S-plus): now have toolbar support with icons to evaluate code in the inferior process or to switch there. This code is experimental and likely to change as XEmacs/Emacs issues get resolved. The toolbar should be enabled if your Emacs displays images, but can be disabled with the variable ess-use-toolbar. Thanks to David Smith from Insightful for the S-plus logo.
- ESS[SAS]: ess-sas-graph-view (*F12*) enhanced; you can specify external file viewers for each graphics file type via the alist ess-sas-graph-view-viewer-alist; also .jpg/.gif are now handled by image-mode on XEmacs, if available, otherwise by graphics primitives as before

Changes/New Features in 5.2.0:

- ESS[BUGS]: new info documentation! now supports interactive processing thanks to Aki Vehtari; new architecture-independent unix support as well as support for BUGS v. 0.5
- ESS[SAS]: convert .log to .sas with ess-sas-transcript; info documentation improved; Local Variable bug fixes; SAS/IML statements/functions now highlighted; files edited remotely by ange-ftp/EFS/tramp are recognized and pressing SUBMIT opens a buffer on the remote host via the local variable ess-sas-shell-buffer-remote-init which defaults to "ssh"; changed the definition of the variable ess-sas-edit-keys-toggle to boolean rather than 0/1; added the function ess-electric-run-semicolon which automatically reverse indents lines containing only "run;"; *C-F1* creates MS RTF portrait from the current buffer; *C-F2* creates MS RTF landscape from the current buffer; *C-F9* opens a SAS DATASET with PROC INSIGHT rather than PROC FSVIEW; "inferior" aliases for SAS batch: *C-c C-r* for submit region, *C-c C-b* for submit buffer, *C-c C-x* for goto .log; *C-c C-y* for goto .lst
- ESS[S]: Pressing underscore ("_") once inserts " <- " (as before); pressing underscore twice inserts a literal underscore. To stop this smart behaviour, add "(ess-toggle-underscore nil)" to your .emacs after ess-site has been loaded; ess-dump-filename-template-proto (new name!) now can be customized successfully (for S language dialects); Support for Imenu has been improved; set ess-imenu-use-S to non-nil to get an "Imenu-S" item on your menubar; ess-help: Now using nice underlines (instead of 'nuke-* ^H_')
- ESS[R]: After (require 'essa-r), *M-x ess-r-var* allows to load numbers from any Emacs buffer into an existing *R* process; *M-x ess-rdired* gives a "directory editor" of R objects; fixed ess-retr-lastvalue-command, i.e. .Last.value bug (thanks to David Brahm)
- ESS: Support for creating new window frames has been added to ESS. Inferior ESS processes can be created in dedicated frames by setting inferior-ess-own-frame to t. ESS help buffers can also open in new frames; see the documentation for ess-help-own-frame for details. (Thanks to Kevin Rodgers for contributing code.)

Changes/New Features in 5.1.24:

- The version number is now correct even inside ESS/Emacs

Changes/New Features in 5.1.23:

- Minor more Makefile clean up.

Changes/New Features in 5.1.22:

- Besides info documentation, PDF and HTML documentation are also provided (instead of built using "make") and available on the web as well; see ESS web page and StatLib
- Now that info documentation is available, the README.* files are no longer supported. However, they are still distributed for what it's worth.
- ESS is now an XEmacs package! See XEmacs Installation HOWTO for details (specifically, items 10-15).
- ESS[SAS]: more user-friendly enhancements for remote SAS batch jobs with Kermit file transfers (LOG and OUTPUT function key features now supported). Multiple shells now supported so you can run SAS on different computers from different buffers by setting the buffer-local variable ess-sas-shell-buffer to unique buffer names.
- Major re-vamping of Makefile/Makeconf.

Changes/New Features in 5.1.21:

- ESS[SAS]: info documentation now available!, see ESS->Help for SAS; *F12* opens GSAS-FILE nearest point for viewing either within emacs, when available, or via an external viewer; more syntax highlighting keywords; more enhancements for remote SAS batch jobs with Kermit; new framework for remote SAS interactive jobs, see ess-remote
- ESS[S]: info documentation now available!, see ESS->Help for the S family
- Makefile: tag now independent of rel; info files made by doc/Makefile and installed in new info sub-directory

Changes/New Features in 5.1.20:

- New 'options()$STERM' in the S dialects (S, S-Plus, R). The S program can determine the environment in which it is currently running. ESS sets the option to 'iESS' or 'ddeESS' when it starts an S language process. We recommend other specific values for S language processes that ESS does not start.
- New 'ess-mouse-me' function, assigned to S-mouse-3 by default. User may click on a word or region and then choose from the menu to display the item, or a summary, or a plot, etc. This feature is still under development.
- GNU Emacs 21.1 is now supported (fixed for S dialects, SAS & BUGS), (some from Stephen Eglen).
- XEmacs 21.x is now supported (fixed w32-using-nt bug)
- XEmacs on Win (NT) is better supported.
- Workaround for bug in Sqpe+6 (S-PLUS 6 for Win).
- should now work even when imenu is not available (for old XEmacsen).
- ESS[SAS]: XEmacs-Imenu fix; *C-TAB* is globalized along with your function-key definitions, if specified; you can specify your SAS library definitions outside of autoexec.sas for ess-sas-data-view with SAS code placed in the variable ess-sas-data-view-libname, also the dataset name is defaulted to the nearest permanent dataset to point; Speedbar support now works for permanent datasets, please ignore first./last.; new font-locking is now the default with more improvements for font-locking PROCs, macro statements, * ; and %* ; comments; you can toggle sas-log-mode with *F10* which will font-lock

your .log (if it isn't too big); submit remote .sas files accessed with ange-ftp, EFS or Tramp (Kermit is experimental) by setting ess-sas-submit-method to 'sh; ess-sas-submit-command and ess-sas-submit-command-options are buffer-local so you can have local file variable sections at the end of your .sas files to request different executables or specify special options and the local file variables are re-read at submit instead of only at file open so that if you make a change it is picked up immediately;

- ESS[BUGS]: font-lock with 'in' fixed.
- for STATA: font-lock bug fixed.
- for Rd mode: `C-c C-v` and 'switch-process' in menu. further, `C-c C-f` prefix (Rd-font) for inserting or surrounding a word by things such as \code{.}, \code{\link{.}}, \emph{.} etc.
- new functions (ess-directory-function) and (ess-narrow-to-defun) ess-directory `<->` default-directory logic (Jeff Mincy).
- Re-organized Makefile and fixed a few bugs.

Changes/New Features in 5.1.19:

- S+6 now supported (Tony Rossini (Unix) and Rich Heiberger (Windows))
- New BUGS support through ESS[BUGS] mode (Rodney Sparapani) Templates assist you in writing .bug and .cmd code (.cmd and .log are replaced by .bmd and .bog to avoid emacs extension collisions). Substitution" parameters facilitate "automagic" generation of data...in" and "init...in" filenames, "const N=" from your data file and "monitor()/stats()" commands. Activated by pressing *F12*.
- Fixes for 'ess-smart-underscore' SAS breakage (Rich Heiberger)
- You can change between PC and Unix, local and global SAS function-key definitions interactively (Rich Heiberger)
- *C-Submit* a highlighted region to SAS batch (Rodney Sparapani)
- New and improved SAS syntax highlighting (Rodney Sparapani) To get the new functionality, set ess-sas-run-make-regexp to nil. Also available in .log files via *F10*.
- Open a permanent SAS dataset for viewing via *F9* (Rodney Sparapani) You must have the library defined in autoexec.sas for it to work.
- User-friendly defaults for 'sas-program', 'ess-sas-batch-pre-command' and 'ess-sas-batch-post-command' as well Customize support for these and other ESS[SAS] variables (Rodney Sparapani)
- 'ess-sas-suffix-2' now defaults to .dat via *F11* (Rodney Sparapani)
- Emacs/XEmacs, Unix/Windows issues collectively handled in ess-emcs.el
- defadvice solves problem of missing *ESS* (thanks to Jeff Mincy)
- Improved manual a bit by including things that were only in 'README'.

Changes/New Features in 5.1.18:

- New 'ess-smart-underscore' function, now assigned to "_" by default. Inserts 'ess-S-assign' (customizable " <- "), unless inside string and comments where plain "_" is used instead. (MM)
- Fixes for longstanding interactive SAS breakage (RMH)

Changes/New Features in 5.1.17:

- Documentation for Windows Installation (Rich Heiberger)
- removal of ess-vars, finalization of customize support (in the sense that there is no more use of ess-vars, but that we need to fix ess-cust) (AJ Rossini)
- Many small (and large) fixes/contributions (MMaechler)
- addition of the "S-equal" variable and provide *M-x ess-add-MM-keys* a way to remap "_" to 'ess-S-assign', typically " <- ", but customizable. (MMaechler)

Changes/New Features in 5.1.16:

- BUG FIXES
- Better SAS support

Changes/New Features in 5.1.15:

- BUG FIXES

Changes/New Features in 5.1.14:

- Yet more fixes to SAS mode, (Rich Heiberger and Rodney Sparapani)
- Customize support (for most Emacsen which support it) (AJRossini)
- ARC and ViSta support out of the box, and fixes for XLispStat (AJRossini)

Changes/New Features in 5.1.13:

- Version numbering finally all depending on the ./VERSION file, thanks to Martin Maechler.
- Yet more fixes to SAS mode, thanks to Rich Heiberger.

Changes/New Features in 5.1.12:

- Splus 5.1 stabilized, thanks to Martin Maechler, Bill Venables, Chuck Taylor, and others.
- More fixes to SAS mode, thanks to Rodney Sparapani and Rich Heiberger.

Changes/New Features in 5.1.11:

- More fixes to Stata mode, thanks to Brendan Halpin.
- fixed bugs in ESS-elsewhere, thanks to many testers
- README.SPLUS4WIN has DETAILED instructions for S-PLUS 2000, thanks to David Brahm.
- Fixes to SAS mode, thanks to Rodney Sparapani

Changes/New Features in 5.1.10:

- More fixes to Stata mode
- primitive generic version of ESS-elsewhere
- Small fixes to SAS/Stata.

Changes/New Features in 5.1.9:

- Stata mode works
- Literate Data Analysis using Noweb works

Changes/New Features in 5.1.8:

- Bug fixes

- R documentation mode defaults changed

Changes/New Features in 5.1.2:

- able to use inferior iESS mode to communicate directly with a running S-Plus 4.x process using the Microsoft DDE protocol. We use the familiar (from Unix ESS) `C-c C-n` and related key sequences to send lines from the S-mode file to the inferior S process. We continue to edit S input files in ESS[S] mode and transcripts of previous S sessions in ESS Transcript mode. All three modes know the S language, syntax, and indentation patterns and provide the syntactic highlighting that eases the programming tasks.

1.3 Authors of and contributors to ESS

The ESS environment is built on the open-source projects of many contributors, dating back nearly 15 years. Doug Bates and Ed Kademan wrote S-mode in 1989 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 process interaction based on Olin Shivers' comint package.

John Sall wrote GNU Emacs macros for SAS source code around 1990. Tom Cook added more functionality creating SAS-mode which was distributed in 1994. Also 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.

- 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 S-mode mailing list.
- Thanks to Frank Ritter for updates from the previous version, the menu code, and invaluable comments on the manual.
- Thanks to Ken'ichi Shibayama for his excellent indenting code, and many comments and suggestions.
- Last but definitely not least, thanks to the many beta testers of the S-mode and ESS mailing lists.

ESS version 5 is being developed and currently maintained by

- A.J. Rossini
- Richard M. Heiberger
- Kurt Hornik
- Martin Maechler
- Rodney A. Sparapani
- Stephen Eglen

1.4 Getting the latest version of ESS

The latest released version of ESS is always available on the web at: ESS web page or StatLib

The latest development version of ESS is available via `https://svn.R-project.org/ESS/`, the ESS Subversion repository. If you have a Subversion client (see `http://subversion.tigris.org/`), you can download the sources using:

```
% svn checkout https://svn.r-project.org/ESS/trunk path
```

which will put the ESS files into directory *path*. Later, within that directory, 'svn update' will bring that directory up to date. Windows-based tools such as TortoiseSVN are also available for downloading the files. Alternatively, you can browse the sources with a web browser at: ESS SVN site. However, please use a subversion client instead to minimize the load when retrieving.

If you remove other versions of ESS from your emacs load-path, you can then use the development version by adding the following to .emacs:

```
(load "/path/to/ess-svn/lisp/ess-site.el")
```

Note that https is required, and that the SSL certificate for the Subversion server of the R project is

```
Certificate information:
 - Hostname: svn.r-project.org
 - Valid: from Jul 16 08:10:01 2004 GMT until Jul 14 08:10:01 2014 GMT
 - Issuer: Department of Mathematics, ETH Zurich, Zurich, Switzerland, CH
 - Fingerprint: c9:5d:eb:f9:f2:56:d1:04:ba:44:61:f8:64:6b:d9:33:3f:93:6e:ad
```

(currently, there is no "trusted certificate"). You can accept this certificate permanently and will not be asked about it anymore.

1.5 How to read this manual

If you need to install ESS, read Chapter 2 [Installation], page 13 for details on what needs to be done before proceeding to the next chapter.

In this manual we use the standard notation 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 or EDIT or ALT key 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 finally, if available, 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. e.g. 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 often referred to as a 'self-documenting' text editor. This applies to ESS in two ways. First, limited 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, a complete list of keybindings that are available in each ESS mode and brief description of that mode is available by typing `C-h m` within an ESS buffer.

Emacs is a versatile editor written in both C and lisp; ESS is written in the Emacs lisp dialect (termed 'elisp') and thus 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 64 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

The following section details those steps necessary to get ESS running on your system.

We now discuss installation, which might happen under Unix or Microsoft Windows. First, we discuss Unix installation. See Section 2.1 [Unix installation], page 13.

For Microsoft Windows Installation please skip to the See Section 2.2 [Microsoft Windows installation], page 14.

2.1 Unix installation

1. `cd` to a directory where you want to install ESS, creating it if necessary. This directory will be referred to below as '`ESSDIR`'.
2. Retrieve the latest version from ESS downloads area to '`ESSDIR`'.
3. Decompress/unarchive the files from the disribution.

   Either, `gunzip < ess-VERSION.tar.gz | tar xf -`,
   or using GNU tar, `tar zxf ess-VERSION.tar.gz`.

   The `tar` command will create the subdirectory '`ess-VERSION`' and install the files there.
4. Edit the file '`ESSDIR/ess-VERSION/lisp/ess-site.el`' as explained in the comments section of that file.
5. Add the line

   ```
   (load "ESSDIR/ess-VERSION/lisp/ess-site")
   ```

   to your user or system installation file (GNU Emacs uses '`$HOME/.emacs`' and XEmacs uses '`$HOME/.xemacs/init.el`' for the user initialization file. GNU Emacs uses '`default.el`' or '`site-init.el`' and XEmacs uses '`site-start.el`' for the system installation file).

   Alternatively, if '`ess-site.el`' is in your current `load-path`, then:

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

   to configure emacs for ESS.
6. That's it! If you are installing just a local copy of ESS for yourself, ESS is now ready to be used. (The remaining step below is for advanced installation.) To edit statistical programs, load the files with the requiste extensions ('`.sas`' for SAS, '`.S`' for S-PLUS, '`.R`' for R, and '`.lsp`' for XLispStat). To start a statistical process within emacs, such as R, type `M-x R`.
7. **(OPTIONAL) READ THIS ITEM THOROUGHLY BEFORE STARTING**:

   If you want to place the compiled files in other locations edit the `LISPDIR`, `INFODIR` and `ETCDIR` entries in Section 1 of '`Makeconf`' in the '`ESSDIR/ess-VERSION`' directory (if you are using XEmacs, then see the XEmacs subsection in Section 1 of '`Makeconf`').

   You can compile those files by:

   ```
   make all
   ```

   When that completes successfully, install the compiled files:

   ```
   make install
   ```

   Note: ESS is no longer available as an XEmacs package. However, ESS will work best if the XEmacs sumo tarball (all XEmacs packages combined) has been installed. For more information on installing XEmacs packages, see Quickstart Package Guide.

2.2 Microsoft Windows installation

For **Microsoft Windows installation**, please follow the next steps: (see separate instructions above for UNIX See Section 2.1 [Unix installation], page 13.

1. `cd` to a directory where you keep emacs lisp files, or create a new directory (for example, '`c:\emacs\`') to hold the distribution. This directory will be referred to below as "the ESS distribution directory".
2. Retrieve the latest zip file ('`ess-VERSION.zip`') from ESS downloads area and store it in the ESS distribution directory. Be aware that http browsers on Windows frequently change the "." and "-" characters in filenames to other punctuation. Please change the names back to their original form.
3. Extract all the files from '`ess-VERSION.zip`' into the ESS distribution directory. In Windows Explorer, you can unpack the archive by double clicking on the folder; you should then see a new folder called '`ess-VERSION`'. Drag that folder into your ESS distribution directory.
4. Add the line

   ```
   (load "C:/emacs/ess-VERSION/lisp/ess-site")
   ```

   to your emacs initialization file. (GNU Emacs uses the filename '`~/.emacs`' and XEmacs uses the filename '`~/.xemacs/init.el`' for the initialization file. The tilde is recognised by emacs as your HOME directory, i.e. the value of your HOME environment variable.) Replace `VERSION` above with the version number of ESS. Remember to use forwardslashes `/` rather than `\\` in your filename.

   After saving your initialization file, ESS is now installed. Start a new emacs and you should be ready to use ESS. For example, to edit statistical programs, load the files with the requisite extensions (".sas" for SAS, ".S" or "s" or "q" or "Q" for S-PLUS, ".r" or ".R" for R, and ".lsp" for XLispStat). One further step is needed if you wish to run statistical processes, see below.
5. To run statistical processes under ESS, Windows users will need to make sure that the directories for the software they will be using is in the PATH environment variable. On Windows 9x, add lines similar to the following to your '`c:\autoexec.bat`' file:

   ```
   path=%PATH%;c:\progra~1\insightful\splus70\cmd
   ```

   On Windows NT/2000/XP, add the directories to the PATH using the `My Computer/Control Panel/System/Advanced/Environment Variables` menu. Note that the directory containing the program is added to the PATH, not the program itself. One such line is needed for each software program. Be sure to use the abbreviation `progra~1` and not the long version with embedded blanks. Use backslashes "\".

   An alternative, for R users, is that rather than adjusting the PATH variable, you can add the following to your emacs initialization file (and restart emacs):

   ```
   (setq inferior-R-program-name "C:/progra~1/R/R-2.2.1/bin/Rterm.exe")
   ```

   This assumes that you have installed R-2.2.1 in the default location. Change the path otherwise to point to other locations.

   To start the S-PLUS [67].x GUI from ESS under emacs:

   1. If you use Cygwin bash as your primary shell, then

```
M-x S
(or M-x S+6).
```

2. If you use the MSDOS prompt window as your primary shell, then

```
M-x S+6-msdos
```

You will then be asked for a pathname ("S starting data directory?"), from which to start the process. The prompt will propose your current directory as the default. ESS will start the S-PLUS GUI. There will be slight delay during which emacs is temporarily frozen. ESS will arrange for communication with the S-PLUS GUI using the DDE protocol. Send lines or regions from the emacs buffer containing your S program (for example, '`myfile.s`') to the S-PLUS Commands Window with the `C-c C-n` or `C-c C-r` keys. (If you are still using S-PLUS 4.x or 2000,\ then use `M-x S+4` or `M-x S+4-msdos`.)

To start an S-PLUS [67].x session inside an emacs buffer—and without the S-PLUS GUI:

```
M-x Sqpe
(or M-x Sqpe+6).
```

This works with both the bash and msdos shells. You will then be asked for a pathname ("S starting data directory?"), from which to start the process. The prompt will propose your current directory as the default. You get Unix-like behavior, in particular the entire transcript is available for emacs-style search commands. Send lines or regions from the emacs buffer containing your S program (for example, '`myfile.s`') to the *S+6* buffer with the `C-c C-n` or `C-c C-r` keys. Interactive graphics are available with Sqpe by using the java library supplied with S-PLUS 6.1 and newer releases. Enter the commands:

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

Graphs can be saved from the `java.graph` device in several formats, but not PostScript. If you need a PostScript file you will need to open a separate `postscript` device. (If you are still using S-PLUS 4.x or 2000, then use `M-x Sqpe+4`.)

To connect to an already running S-PLUS GUI (started, for example, from the S-PLUS icon):

```
M-x S+6-existing
```

or

```
M-x S+6-msdos-existing
```

You will then be asked for a pathname ("S starting data directory?"), from which to start the process. The prompt will propose your current directory as the default. ESS will arrange for communication with the already running S-PLUS GUI using the DDE protocol. Send lines or regions from the emacs buffer containing your S program (for example, '`myfile.s`') to the S-PLUS Commands Window with the `C-c C-n` or `C-c C-r` keys. (If you are still using S-PLUS 4.x or 2000, then use `M-x S+4-existing` or `M-x S+4-msdos-existing`.)

If you wish to run R, you can start it with:

```
M-x R
```

XLispStat can not currently be run with

```
M-x XLS
```

Hopefully, this will change. However, you can still edit with emacs, and cut and paste the results into the XLispStat *Listener* Window under Microsoft Windows.

6. That's it!

2.3 Requirements

ESS has been tested with

- R >=0.49
- S-PLUS 3.3-4, 4.5, 5.0-1, 6.0-2, 7.0, 2000
- S4
- SAS >=6.12
- BUGS 0.5, 0.603
- Stata >=6.0
- XLispStat >=3.50

on the following platforms

- Linux (all)
- Solaris/SunOS (all)
- Microsoft Windows 95/98/NT/2000/XP (R, SPLUS 4.5/6.0-2/7.0/2000, SAS and BUGS)
- Apple Mac OS (SAS for OS 9 and R for OS X)

with the following versions of emacs

- GNU Emacs 20.3-7, 21.1, 21.3-4, 22.0.50-pretesting
- XEmacs 21.0, 21.1.13-14, 21.4.0-8, 21.4.9-13[1], 21.4.14-15, 21.4.17-18, 21.5.23

[1] require the files.el patch to revert-buffer for the Local Variables updating problem

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 S from within emacs. The emacs convention is to name such processes running under its control as 'inferior processes'. This term can be slightly misleading, in which case these processes can be thought of '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 S session on Unix or on Windows when you use the Cygwin bash shell, simply type *M-x S RET*.

To start an S session on Windows when you use the MSDOS prompt shell, simply type *M-x S+6-msdos RET*.

S will then (by default) ask the question

```
S starting data directory?
```

Enter the name of the directory you wish to start S from (that is, the directory you would have `cd`'d to before starting S from the shell). This directory should have a '`.Data`' subdirectory.

You will then be popped into a buffer with name '`*S*`' 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 ESS process simultaneously in the same session. Each process has a name and a number; the initial process (process 1) is simply named (using S-PLUS as an example) '`S+3:1`'. The name of the process is shown in the mode line in square brackets (for example, '`[S+3:2]`'); this is useful if the process buffer is renamed. Without a prefix argument, *M-x S* starts a new ESS process, using the first available process number. With a prefix argument (for R), *C-u M-x R* allows for the specification of command line options.

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 S processes. This is a good command to consider binding to a global key.

3.3 ESS processes on Remote Computers

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 from a telnet or ssh buffer and then connect ESS to that buffer.

1. Start a new telnet or ssh buffer and connect to the remote computer (e.g. use '`M-x telnet`' or '`M-x ssh`'; ssh.el is available at ftp://ftp.splode.com/pub/users/friedman/emacs-lisp/ssl

2. Start the ESS process on the remote machine, for example with one of the commands '`Splus`', or '`R`', or '`sas -stdio`'.
3. Enter the ESS command '`M-x ess-remote`'. You will be prompted for a program name. Enter '`sp6`' or '`r`' or '`sas`' or another valid name. Your telnet process is now known to ESS. All the usual ESS commands ('`C-c C-n`' and its relatives) now work with the S 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 12.3.2 [X11], page 60, 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 10.5 [ESS(SAS)–Function keys for batch processing], page 51, 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.

We have two older commands, now deprecated, for accessing ESS processes on remote computers. See Section 3.4 [S+elsewhere and ESS-elsewhere], page 18.

3.4 S+elsewhere and ESS-elsewhere

These commands are now deprecated. We recommend '`ess-remote`'. We have two versions of the elsewhere function. '`S+elsewhere`' is specific for the S-Plus program. The more general function '`ESS-elsewhere`' is not as stable.

1. Enter '`M-x S+elsewhere`'. You will be prompted for a starting directory. I usually give it my project directory on the local machine, say '`~myname/myproject/`'

   Or enter '`M-x ESS-elsewhere`'. You will be prompted for an ESS program and for a starting directory. I usually give it my project directory on the local machine, say '`~myname/myproject/`'
2. The '`*S+3*`' buffer will appear with a prompt from the local operating system (the unix prompt on a unix workstation or with cygwin bash on a PC, or the msdos prompt on a PC without bash). emacs may freeze because the cursor is at the wrong place.

Unfreeze it with '`C-g`' then move the cursor to the end with '`M->`'. With '`S+elsewhere`' the buffer name is based on the name of the ESS program.

3. Enter '`telnet myname@other.machine`' (or '`ssh myname@other.machine`'). You will be prompted for your password on the remote machine. Use '`M-x send-invisible`' before typing the password itself.
4. Before starting the ESS process, type '`stty -echo nl`' at the unix prompt. The '`-echo`' turns off the echo, the '`nl`' turns off the newline that you see as '`^M`'.
5. You are now talking to the unix prompt on the other machine in the '`*S+3*`' buffer. cd into the directory for the current project and start the ESS process by entering '`Splus`' or '`R`' or '`sas -stdio`' as appropriate. If you can login remotely to your Windows 2000, then you should be able to run '`Sqpe`' on the Windows machine. I haven't tested this and noone has reported their tests to me. You will not be able to run the GUI through this text-only connection.
6. Once you get the S or R or SAS prompt, then you are completely connected. All the '`C-c C-n`' and related commands work correctly in sending commands from '`myfile.s`' or '`myfile.r`' on the PC to the '`*S+3*`' buffer running the S or R or SAS program on the remote machine.
7. Graphics on the remote machine works fine. If you run the X window system on the remote unix machine you should be able to display them in '`xfree86`' on your PC. If you don't run X11, then you can write graphics to the postscript device and copy it to your PC with dired and display it with ghostscript.

3.5 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 value of the variable `ess-directory` is used as the starting directory. The default value for this variable is your home directory. If `ess-ask-for-ess-directory` has a non-`nil` value (as it does by default) then the value of `ess-directory` 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 S sessions, 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`, as 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 (and you should give it a file name ending in '`.St`'); 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 4.4.4 [Saving transcripts], page 25.)

Once these questions are answered (if they are asked at all) the S process itself is started by calling the program name specified in the variable `inferior-ess-program`. If you need

to pass any arguments to this program, they may be specified in the variable `inferior-`*S_program_name*`-args` (e.g. if `inferior-ess-program` is `"S+"` then the variable to set is `inferior-S+-args`. It is not normally necessary to pass arguments to the S 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 S process to appear in a separate variable, 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 S interpreter. This is achieved by running the S process from within an Emacs buffer, so that the Emacs editing commands are available to correct mistakes in commands, etc. The features of Inferior S 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 S objects and a number of 'hot keys' for commonly-used S 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:

- *RET* (`inferior-ess-send-input`)
  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.

ESS 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-c 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 Completion of object names

In the process buffer, the ⟨TAB⟩ key is for completion, similar to that provided by Shell Mode for filenames. In Inferior S mode, pressing the ⟨TAB⟩ key when the cursor is following the first few characters of an object name *completes* the object name; if the cursor is following a file name *TAB* completes the file name.

- *TAB* (`comint-dynamic-complete`)
  Complete the S 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 `tcsh` except that completion is performed over all known S object names instead of file names. ESS maintains a list of all objects known to S 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 (if your version of S supports them).

For example, consider the three functions (available in Splus version 3.0) called `binomplot()`, `binom.test()` and `binomial()`. Typing *`bin TAB`* after the S prompt 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-?`*.

- *`M-?`* (`ess-list-object-completions`)
  List all possible completions of the object name at point.

ESS also provides completion over the components of named lists accessed using the '`$`' notation, to any level of nested lists. This feature is particularly useful for checking what components of a list object exist while partway through entering a command: simply type the object name and '`$`' and press *`TAB`* to see the names of existing list components for that object.

Completion is also provided over file names, which is particularly useful when using S functions such as `get()` or `scan()` which require fully expanded file names. Whenever the cursor is within an S string, pressing *`TAB`* completes the file name before point, and also expands any '`~`' or environment variable references.

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 4.6 [History expansion], page 26.

4.3 Completion details

ESS automatically keeps track of any objects added or deleted to the system (such as new objects created, or directories added to the search list) to make completion as accurate as possible. Whenever ESS notices that search list has changed [1] when you attach a directory or data frame, the objects associated with it immediately become available for a completion; when it is detached completion is no longer available on those objects.

To maintain a list of accessible objects for completion, ESS needs to determine which objects are contained in each directory or data frame on the search list. This is done at the start of each S session, by running the `objects()` command on every element of the search list.

Efficiency in completion is gained by maintaining a cache of objects currently known to S; 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.

[1] The variable `ess-change-sp-regexp` is a regular expression matching commands which change the search list. You will need to modify this variable if you have defined custom commands (other than `attach`, `detach`, `collection` or `library`) which modify the search list.

4.4 Manipulating the transcript

Most of the time, the cursor spends most of 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*.

4.4.1 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 a longish 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 command

- `C-c C-e` (`comint-show-maximum-output`)
  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`.)

If the first part of the output is still obscured, use

- `C-c C-r` (`comint-show-output`)
  Moves cursor to the previous command line and and places it at the top of the window.

to view it. Finally, if you want to discard the last command output altogether, use

- `C-c C-o` (`comint-kill-output`)
  Deletes everything from the last command to the current prompt.

to delete it. Use this command judiciously to keep your transcript to a more manageable size.

4.4.2 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-input)
  Moves point to the preceding command in the process buffer.
- *C-c C-n* (comint-next-input)
  Moves point to the next command 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.

4.4.3 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 three 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 signalled). Note all three commands involve the ⟨RETURN⟩ key.

- *RET* (inferior-ess-send-input)
  Copy the command under the cursor to the current command line, and execute it.
- *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.
- *M-RET* (ess-transcript-send-command-and-move)
  Copy the command under the cursor to the current command line, and execute it. Moves the cursor to the following command.

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 submitting it.

These two commands leave the cursor at the new command line, allowing you to continue with interactive use of S. If you wish to resubmit a series of commands from the transcript, consider using *M-RET* instead, which leaves the cursor at the command line following the one you re-submitted. Thus by using *M-RET* repeatedly, you can re-submit a whole series of commands.

These commands work even if 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 4.5 [Command History], page 25.

4.4.4 Keeping a record of your S session

To keep a record of your S 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 S 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 S by setting the variable `ess-ask-about-transfile` to `t`; see Section 3.5 [Customizing startup], page 19. We recommend you save your transcripts with filenames that end in '`.St`'. There is a special mode (ESS transcript mode — see Chapter 5 [Transcript Mode], page 30) for editing transcript files which is automatically selected for files with this suffix.

S 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-e* (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.

Finally, if you intend to produce S source code (suitable for using with `source()` or inclusion in an S function) from a transcript, then the command *M-x ess-transcript-clean-region* may be of use. This command works in any Emacs buffer, and removes all prompts and command output from an ESS transcript within the current region, leaving only the commands. Don't forget to remove any erroneous commands first!

4.5 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 50 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-previous-matching-input`)
  Prompt for a regular expression, and search backwards through the input history for a command matching the expression.
- *M-s* (`comint-next-matching-input`)
  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:

- *A-M-r* (`comint-previous-matching-input-from-input`)
  Select the previous command in the history which matches the string typed so far.
- *A-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 *A-M-r* and *RET*. (Note: you may not have an ⟨ALT⟩ key on your keyboard, in which case it may be a good idea to bind these commands to some other keys.)

See section "Shell History Ring" in *The GNU Emacs Reference Manual*, for a more detailed discussion of the history mechanism.

4.6 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 S, 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 S 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*.

4.7 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 variable `ess-execute-in-process-buffer` which, 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, i.e. 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:

- *C-c C-x* (`ess-execute-objects`)
  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). A quick way to see what objects are in your working directory.

- *C-c C-s* (`ess-execute-search`)
  Sends the `search()` command to the ESS process.
- *C-c C-e* (`ess-execute`)
  Prompt for an ESS expression, and evaluate it.

`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:

- *C-c C-a* (`ess-execute-attach`)
  Prompts for a directory to attach to the ESS process with the `attach()` command. If a numeric prefix argument is given it is used as the position on the search list to attach the directory; otherwise the S default of 2 is used. The `attach()` command actually executed appears in the process buffer.
- *C-c C-l* (`ess-load-file`)
  Prompts for a file 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 *C-x `* (`ess-parse-errors`). See Section 7.3 [Error Checking], page 38, for more details.
- *C-c C-v* (`ess-display-help-on-object`)
  Pops up a help buffer for an S object or function. See Chapter 9 [Help], page 46 for more details.
- *C-c C-q* (`ess-quit`)
  Sends the `q()` command to the ESS process (or `(exit)` to the **XLS** process), and cleans up any temporary buffers (such as help buffers or edit buffers) you may have created along the way. Use this command when you have finished your S session instead of simply typing `q()` yourself, otherwise you will need to issue the command *M-x ess-cleanup* command 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.

4.8 Is the Statistical Process running under ESS?

For the S languages (S, S-Plus, R) 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 S 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 S language program is not under the control of ESS.

4.9 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 principle applies.) Within your R session, for example, 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 S function `page(x)` will also show a text representation of the object `x` in a temporary Emacs buffer.

4.10 Other commands provided by inferior-ESS

The following commands are also provided in the process buffer:

- *C-c C-c* (`comint-interrupt-subjob`)
  Sends a Control-C signal to the ESS process. This has the effect of aborting the current command.
- *C-c C-z* (`ess-abort`)
  Sends a STOP signal to the ESS process, killing it immediately. It's not a good idea to use this, in general: Neither `q()` nor `.Last` will be executed and device drivers will not finish cleanly. This command is provided as a safety to `comint-stop-subjob`, which is usually bound to *C-c C-z*. If you want to quit from S, use *C-c C-q* (`ess-quit`) instead.

- *C-c C-d* (`ess-dump-object-into-edit-buffer`)
  Prompts for an object to be edited in an edit buffer. See Chapter 7 [Editing], page 37.

Other commands available in Inferior S mode are discussed in section "Shell Mode" in *The Gnu Emacs Reference Manual*.

5 Manipulating saved transcript files

Inferior S 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 '`.St`'. The most obvious use for a transcript file is as a static record of the actions you have performed in a particular S 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 '`.St`' into Emacs, it is placed in S Transcript Mode. Transcript Mode is similar to Inferior S mode (see Chapter 4 [Entering commands], page 21): 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*.

5.1 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:

- *RET* (`ess-transcript-send-command`)
  Send the current command line to the ESS process, and execute it.
- *C-c RET* (`ess-transcript-copy-command`)
  Copy the current command to the ESS process, and switch to the ESS process buffer (ready to edit the copied command).
- *M-RET* (`ess-transcript-send-command-and-move`)
  Send 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 these commands are similar to those on the same keys in Inferior S 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.

5.2 Cleaning transcript files

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

- *C-c C-w* (`ess-transcript-clean-region`)
  Deletes all prompts and command output in the region, leaving only the commands themselves.

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 Section 7.4 [Evaluating code], page 38) using the code evaluation commands from S mode, also available in S Transcript Mode.

6 ESS for the S family

6.1 ESS[S]–Editing files

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

- proper indenting, generated by both [Tab] and [Return].
- color and font choices based on syntax.
- ability to send the contents of an entire buffer, a highlighted region, an S function, or a single line to an inferior S process, if one is currently running.
- ability to switch between processes which would be the target of the buffer (for the above).
- The ability to request help from an S process for variables and functions, and to have the results sent into a separate buffer.
- completion of object names and file names.

ESS[S] mode should be automatically turned on when loading a file with the suffices found in ess-site (*.R, *.S, *.s, etc). However, one will have to start up an inferior process to take advantage of the interactive features.

6.2 iESS[S]–Inferior ESS processes

iESS (inferior ESS) is the mode for interfacing with active statistical processes (programs). This mode handles:

- proper indenting, generated by both [Tab] and [Return].
- color and font highlighting based on syntax.
- ability to resubmit the contents of a multi-line command to the executing process with a single keystroke [RET].
- The ability to request help from the current process for variables and functions, and to have the results sent into a separate buffer.
- completion of object names and file names.
- interactive history mechanism.
- transcript recording and editing.

To start up iESS mode, use:

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

(for S-PLUS 3.x, S4, S+5, S+6 or S+7, and R, respectively. This assumes that you have access to each). Usually the site will have defined one of these programs (by default S+6) to the simpler name:

M-x S

In the (rare) case that you wish to pass command line arguments to the starting S+6 process, set the variable `inferior-Splus-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. To enter a command line argument, call R using a "prefix argument", by

C-u M-x R

and when ESS prompts for "Starting Args ? ", enter (for example):

`--vanilla`

Then that R process will be started up using `R --vanilla`. If you wish to always call R with certain arguments, set the variable `inferior-R-args` accordingly.

If you have other versions of R or S-Plus available on the system, ESS is also able to start those versions. How this works depend on which OS you are using:

R on Unix systems: 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" or "R-2". 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 defuns have been created for starting different versions of R, type *M-x R-* and then hit [Tab]. You will then see if any defuns for particular versions of R have been created. 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 defuns have been created for starting different versions of R, type *M-x rw* and then hit [Tab]. You will then see if any defuns for particular versions of R have been created. These other versions of R can also be started from the "ESS->Start Process->Other" menu.

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 defuns have been created for starting different versions of Splus, type *M-x Splus* and then hit [Tab]. You will then see if any defuns for particular versions of Splus have been created. These other versions of Splus can also be started from the "ESS->Start Process->Other" menu.

A second mechanim 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-s-versions` or `ess-s-versions-list`, you should put them in your .emacs 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 defuns have been created for starting different versions of S-Plus, type *`M-x spl`* and then hit [Tab]. You will then see if any defuns for particular versions of S-Plus have been created. These other versions of S-Plus can also be started from the "ESS->Start Process->Other" menu.

6.3 ESS-help–assistance with viewing help

ESS has built-in facilities for viewing help files from S. See Chapter 9 [Help], page 46.

6.4 Philosophies for using ESS[S]

The first is preferred, and configured for. The second one can be retrieved again, by changing emacs variables.

1: (preferred by the current group of developers): The source code is real. The 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: (older version): S 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 file:

```
(setq ess-keep-dump-files 'nil)
(setq ess-delete-dump-files t)
(setq ess-mode-silently-save nil)
```

The second saves a small amount of disk space. The first allows for better portability as well as external version control for code.

6.5 Scenarios for use (possibilities–based on actual usage)

We present some basic suggestions for using ESS to interact with S. 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 (source code is real)
Load the file you want to work with
C-x C-f myfile.s

Edit as appropriate, and then start up S-PLUS 3.x
M-x S+3

```
## A new buffer *S+3:1* will appear.  Splus will have been started
## in this buffer.  The buffer is in iESS [S+3:1] mode.

## Split the screen and go back to the file editing buffer.
C-x 2 C-x b myfile.s

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

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

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

## Save the entire *S+3:1* interaction buffer with C-c C-s.  You
## will be prompted for a file name.  The recommended name is
## myfile.St.  With the *.St 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 up S-PLUS 3.x in a process buffer (this will be *S+3:1*)
M-x S+3

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

## edit program, functions, and code in myfile.s, and send revised
## functions to S 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 C-l

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

3:
```
## Program revision example (S object is real)

## Start up S-PLUS 3.x in a process buffer (this will be *S+3:1*)
```

```
M-x S+3

## Dump an existing S object my.function into a buffer to work with
C-c C-d my.function
## a new buffer named yourloginname.my.function.S will be created with
## an editable copy of the object.  The buffer is associated with the
## pathname /tmp/yourloginname.my.function.S and will amlost certainly not
## exist after you log off.

## enter program, functions, and code into work buffer, and send
## entire contents to S-PLUS when ready
C-c C-b

## Go to *S+3:1* buffer, which is the process buffer, and examine
## the results.
C-c C-y
## The sequence C-c C-y is a shortcut for:  C-x b *S+3:1*

## Return to the work buffer (may/may not be prefixed)
C-x C-b yourloginname.my.function.S
## Fix the function that didn't work, and resubmit by placing the
## cursor somewhere in the function and
C-c C-f
## Or you could've selected a region (using the mouse, or keyboard
## via setting point/mark) and
C-c C-r
## Or you could step through, line by line, using
C-c C-n
## Or just send a single line (without moving to the next) using
C-c C-j
## To fix that error in syntax for the "rchisq" command, get help
## by
C-c C-v rchisq


4:   Data Analysis (S object is real)
## Start up S-PLUS 3.x, in a process buffer (this will be *S+3:1*)
M-x S+3

## Work in the process buffer.  When you find an object that needs
## to be changed (this could be a data frame, or a variable, or a
## function), dump it to a buffer:
C-c C-d my.cool.function

## Edit the function as appropriate, and dump back in to the
## process buffer
C-c C-b

## Return to the S-PLUS process buffer
```

```
C-c C-y
## Continue working.

## When you need help, use
C-c C-v rchisq
## instead of entering:   help("rchisq")
```

6.6 Customization Examples and Solutions to Problems

1. Suppose that you are primarily an SPLUS 3.4 user, occasionally using S version 4, and sick and tired of the buffer-name *S+3* we've stuck you with. Simply edit the "ess-dialect" alist entry in the essd-sp3.el and essd-s4.el files to be "S" instead of "S4" and "S+3". This will ensure that all the inferior process buffer names are "*S*".

2. Suppose that you WANT to have the first buffer name indexed by ":1", in the same manner as your S-PLUS processes 2,3,4, and 5 (for you heavy simulation people). Then uncomment the line in ess-site (or add after your (require 'ess-site) or (load "ess-site") command in your .emacs file, the line:

```
(setq ess-plain-first-buffername nil)
```

)

3. Fontlocking sometimes fails to behave nicely upon errors. When Splus dumps, a mis-matched " (double-quote) can result in the wrong font-lock face being used for the remainder of the buffer.

Solution: add a " at the end of the "Dumped..." statement, to revert the font-lock face back to normal.

7 Editing S functions

ESS provides facilities for editing S objects within your Emacs session. Most editing is performed on S 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 S functions. Error-checking is performed when S code is loaded into the ESS process.

7.1 Creating or modifying S objects

To edit an S object, type

- *C-c C-d* (`ess-dump-object-into-edit-buffer`)
  Edit an S object in its own edit buffer.

from within the ESS process buffer (`*S*`). 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 S source code. Commands are provided to intelligently indent S code, evaluate portions of S code and to move around S code constructs.

Note: when you dump a file with *C-c 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 7.7 [Source Files], page 41) 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 S 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 S'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-d*.

7.2 Loading source files into the ESS process

The best way to get information — particularly function definitions — into S is to load them in as source file, using S's `source` function. You have already seen how to create source files using *C-c C-d*; ESS provides a complementary command for loading source files (even files not created with ESS!) into the ESS process:

- *C-c C-l* (`ess-load-file`)
  Loads a file into the ESS process using `source()`.

After typing *C-c C-l* you will prompted for the name of the file to load into S; 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-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 S 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 Sending code to the ESS process

Other commands are also available for evaluating portions of code in the S 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-p` whose default is `nil` (display output only). Passing a prefix argument (*C-u*) to any of the following commands, however, reverses the meaning of `ess-eval-visibly-p` for that command only — for example *C-u C-c C-j* echoes the current line of S (or other) code in the ESS process buffer, followed by its output. This method of evaluation is an alternative to S's `source()` function when you want the input as well as the output to be displayed. (You can sort of do this with `source()` when the option `echo=T` is set, except that prompts do not get displayed. ESS puts prompts in the right places.) The commands for evaluating code are:

- *C-c C-j* (`ess-eval-line`)
  Send the line containing point to the ESS process.
- *C-c M-j* (`ess-eval-line-and-go`)
  As above, but returns you to the ESS process buffer as well.
- *C-c C-f* or *ESC C-x* (aka *M-C-x*) (`ess-eval-function`)
  Send the S function containing point to the ESS process.
- *C-c M-f* (`ess-eval-function-and-go`)
  As above, but returns you to the ESS process buffer as well.
- *C-c C-r* (`ess-eval-region`)
  Send the text between point and mark to the ESS process.

- *C-c M-r* (ess-eval-region-and-go)
  As above, but returns you to the ESS process buffer as well.
- *C-c C-b* (ess-eval-buffer)
  Send the contents of the edit buffer to the ESS process.
- *C-c M-b* (ess-eval-buffer-and-go)
  As above, but returns you to the ESS process buffer as well.
- *C-c C-n* (ess-eval-line-and-step)
  Sends the current line to the ESS process, echoing it in the process buffer, and moves point to the next line. Useful when debugging for stepping through your code.

It should be stressed once again that these ess-eval- commands should only be used for evaluating small portions of code for debugging purposes, or for generating transcripts from source files. When editing S functions, *C-c C-l* is the command to use to update the function's value. In particular, ess-eval-buffer is now largely obsolete.

One final command is provided for spot-evaluations of S code:

C-c C-t (ess-execute-in-tb)
Prompt for an S expression and evaluate it. Displays result in a temporary buffer.

This is useful for quick calculations, etc.

All the above commands are useful for evaluating small amounts of code and observing the results in the process buffer. 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-scroll-to-bottom-on-output to 'others or t.

7.5 Indenting and formatting S code

ESS now provides a sophisticated mechanism for indenting S source code (thanks to Ken'ichi Shibayama). 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 if 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-fancy-comments nil) to your '.emacs' file.

The indentation commands provided by ESS are:

- *TAB* (`ess-indent-command`)
  Indents the current line as S code. If a prefix argument is given, all following lines which are part of the same (compound) expression are indented by the same amount (but relative indents are preserved).
- *RET* (`newline-and-indent`)
  LFD (*newline-and-indent*)
  Insert a newline, and indent the next line. (Note that most keyboards nowadays do not have a ⟨LINEFEED⟩ key, but *C-j* is equivalent.)
- *ESC C-q* aka *M-C-q* aka *C-M-q* (`ess-indent-exp`)
  Indents each line in the S (compound) expression which follows point. Very useful for beautifying your S code.
- `{` and `}` (`ess-electric-brace`)
  The braces automatically indent to the correct position when typed.
- *M-;* (`indent-for-comment`)
  Indents an existing comment line appropriately, or inserts an appropriate comment marker.
- *M-x ess-set-style*
  Set the formatting style in this buffer to be one of the predefined styles: `GNU`, `BSD`, `K&R`, `CLB`, and `C++`. The `DEFAULT` style uses the default values for the indenting variables (unless they have been modified in your '`.emacs`' file.) This command causes all of the formatting variables to be buffer-local.

7.6 Commands for motion, completion and more

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

- *ESC C-a* aka *C-M-a* (`ess-beginning-of-function`)
  Moves point to the beginning of the function containing point.
- *ESC C-e* aka *C-M-e* (`ess-end-of-function`)
  Moves point to the end of the function containing point.
- *ESC C-h* aka *C-M-h* (`ess-mark-function`)
  Places point at the beginning of the S 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: *M-TAB* completes file names and *M-?* lists file completions. Since ⟨TAB⟩ is used for indentation in the edit buffer, object completion is now performed with *C-c TAB*. Note however that completion is only provided over globally known S objects (such as system functions) — it will *not* work for arguments to functions or other variables local to the function you are editing.

Finally, two commands are provided for returning to the ESS process buffer:

- *C-c C-z* (`ess-switch-to-end-of-ESS`)
  Returns you to the ESS process buffer, placing point at the end of the buffer.

- *C-c C-y* (ess-switch-to-ESS)
  Also returns to to the ESS process buffer, but leaves point where it was.

In addition some commands available in the process buffer are also available in the edit buffer. You can still read help files with *C-c C-v*, edit another function with *C-c C-d* and of course *C-c C-l* can be used to load a source file into S. See Section 4.10 [Other], page 29, for more details on these commands.

7.7 Maintaining S 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-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 S function.

ess-delete-dump-files User Option

If non-nil, dump files created with C-c C-d are deleted immediately after they are created by the ess-process.

Since immediately after S dumps an object's definition to a disk file the source code on disk corresponds exactly to S'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 S 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 S's dump function; it doesn't apply to source files which already exist. The default value is t.

ess-keep-dump-files User Option

Option controlling what to do with the dump file after an object has been successfully loaded into S. Valid values are nil (always delete), ask (always ask whether to delete), check (delete files generated with *C-c C-d* in this Emacs session, otherwise ask — this is the default) and t (never delete). This variable is buffer-local.

After an object has been successfully (i.e. without error) loaded back into S with *C-c C-l*, the disk file again corresponds exactly (well, almost — see below) to S'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 RCS 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-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 S object itself. Another useful feature is that you may format the code in any fashion you

please without S 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-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-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-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`.

7.8 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 '`.S`'. Thus if user `joe` dumps the object `myfun` the dump file will have name '`joe.myfun.S`'. 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 S 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 '`~/S-source`'. This could be achieved by including the following line in your '`.emacs`' file:

```
(setq ess-source-directory (expand-file-name "~/S-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 S working directory during an S 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 S 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 Editing R documentation files

ESS also provides support for editing *R documentation* ("Rd") files. R objects are documented in files written in Rd format, 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.

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 9 [Help], page 46) 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. E.g., *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`
: Hook to be run when Rd mode is entered.

`Rd-indent-level`
: The indentation of Rd code with respect to containing blocks. Default is 2.

`Rd-to-help-command`
: 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)))
```

9 Reading help files

ESS provides an easy-to-use facility for reading S 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 'S 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 S help mode.
- *h* (`ess-display-help-on-object`)
  Pop up a help buffer for a different object

Paging commands:

- *b* or *DEL* (`scroll-down`)
  Move one page backwards through the help file.
- *SPC* (`scroll-up`)
  Move one page forwards through the help file.
- `>` (`beginning-of-buffer`) and `<` (`end-of-buffer`)
  Move to the beginning and end of the help file, respectively.

Section-based motion commands:

- *n* (`ess-skip-to-next-section`) and *p* (`ess-skip-to-previous-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:

  'a' ARGUMENTS:

  'b' BACKGROUND:

  'B' BUGS:

  'd' DETAILS:

  'D' DESCRIPTION:

  'e' EXAMPLES:

  'n' NOTE:

  'o' OPTIONAL ARGUMENTS:

  'r' REQUIRED ARGUMENTS:

  'R' REFERENCES:

  's' SIDE EFFECTS:

‘s’ SEE ALSO:

‘u’ USAGE:

‘v’ VALUE:

‘<’ Jumps to beginning of file

‘>’ Jumps to end of file

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

Miscellaneous:

- *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.
- */* (isearch-forward)
  Same as *C-s*.

Quit commands:

- *q* (ess-switch-to-end-of-ESS)
  Returns to the ESS process buffer in another window, leaving the help window visible.
- *k* (kill-buffer)
  Kills the help buffer.
- *x* (ess-kill-buffer-and-go)
  Return to the ESS process, killing this help buffer.

In addition, all of the ESS commands available in the edit buffers are also available in S help mode (see Section 7.1 [Edit buffer], page 37). 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.

10 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.

10.1 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.

10.2 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 10.6 [iESS(SAS)–Interactive SAS processes], page 54.)

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.

10.3 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 equivalent to uncommenting the following line in '`ess-site.el`':

```
(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 uncommenting the following line in '`ess-site.el`':

```
(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`.

10.4 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 Mac OS X, 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⟩)

10.5 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, uncomment **ONLY ONE** of the following lines in your '`ess-site.el`'. ESS[SAS] function keys are available in ESS[SAS] if you uncomment either 2 or 3 and in all modes if you uncomment 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)
```

10.6 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 10.7 [iESS(SAS)–Common problems], page 55.

10.7 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.

10.8 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;
```

10.9 ESS(SAS)–Windows

- iESS[SAS] does not work on Windows. See Section 10.7 [iESS(SAS)–Common problems], page 55.
- ESS[SAS] mode for editing SAS language files works very well. See Section 10.2 [ESS(SAS)–Editing files], page 48.
- There are two execution options for SAS on Windows. You can use batch. See Section 10.4 [ESS(SAS)–Batch SAS processes], page 49.

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

11 ESS for BUGS

ESS[BUGS] was designed for use with BUGS software. It was developed by Rodney A. Sparapani and has some similarities with ESS[SAS]. ESS facilitates BUGS batch with ESS[BUGS], the mode for files with the .bug extension. ESS 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.

11.1 ESS[BUGS]–Model files

Model files (with the .bug extension) are edited in ESS[BUGS] mode. Two keys are bound for your use in ESS[BUGS], F2 and F12. F2 performs the same action as it does in ESS[SAS], See Section 10.5 [ESS(SAS)–Function keys for batch processing], page 51. F12 performs the function ess-bugs-next-action which you will use a lot. Pressing F12 in an empty buffer for a model file will produce a template for you.

ESS[BUGS] supports "replacement" variables. These variables are created as part of the template, i.e. with the first press of F12 in an empty buffer. They are named by all capitals and start with '%': %N, %DATA, %INIT, %MONITOR and %STATS. When you are finished editing your model file, pressing F12 will perform the necessary replacements and build your command file for you.

The %DATA variable appears in the line 'data in "%DATA";'. On the second press of F12, %DATA will be replaced by the model file name except it will have the .dat extension. If your data file is named something else, then change %DATA in the template to the appropriate file name and no replacement will occur.

The %INIT variable appears in the line 'inits in "%INIT";'. On the second press of F12, %INIT will be replaced by the model file name except it will have the .in extension. If your model will be generating it's own initial values, place a comment character, #, at the beginning of the line. Or, if your init file is named something else, then change %INIT in the template to the appropriate file name.

The %N variable appears in the line 'const N = 0;#%N'. Although it is commented, it is still active. Notice that later on in the template you have the line 'for (i in 1:N)'. The BUGS constant N is the number of rows in your data file. When you press F12, the data file is read and the number of lines are counted (after %DATA is resolved, if necessary). The number of lines replace the zero in the 'const N = 0' statement.

The %MONITOR variable appears on a line by itself. Although it is commented, it is still active. This line is a list of variables that you want monitored. When you press F12, the appropriate statements are created in the command file to monitor the list of variables. If the line is blank, then the list is populated with the variables from the 'var' statement.

The %STATS variable is similar to the %MONITOR variable. It is a list of variables for which summary statistics will be calculated. When you press F12, the appropriate statements will be generated in your command file.

Please note that the %DATA and %INIT variables are only replaced on the second press of F12, but the actions for %N, %MONITOR and %STATS are performed on each press of F12 if you re-visit the model file.

11.2 ESS[BUGS]–Command files

To avoid extension name collision, .bmd is used for BUGS command files. When you have finished editing your model file and press F12, a command file is created if one does not already exist. However, the command file was created, it recognizes two "replacement" variables: %MONITOR and %STATS.

Two %MONITOR variables appears on lines by themselves. Although they are commented, they are still active. Between them appears the necessary statements to monitor the list of variables specified in the model file. The behavior of the %STATS variable is similar.

When you are finished editing your command file, pressing F12 again will submit your command file as a batch job. Batch scripts are provided for both DOS and Unix in the etc sub-directory of the ESS distribution. The DOS script is called "BACKBUGS.BAT" and the Unix script is "backbugs". These scripts allow you to change the number of bins to use in the Griddy algorithm (Metropolis sampling). That is handled by the variable ess-bugs-default-bins which defaults to 32.

11.3 ESS[BUGS]–Log files

To avoid extension name collision, .bog is used for BUGS log files. The BUGS batch script provided with ESS creates the .bog file from the .log file when the batch process completes. If you need to look at the .log file while the batch process is running, it will not appear in ESS[BUGS] mode unless you modify the auto-mode-alist variable. If you have done so, then you may find F2 useful to refresh the .log if the batch process over-writes or appends it.

12 Other features of ESS

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

12.1 Syntactic highlighting of buffers

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

To activate highlighting, you need to turn on Font Lock mode in the appropriate buffers. This can be done on a per-buffer basis with *M-x font-lock-mode*, or may be done by adding `turn-on-font-lock` to `inferior-ess-mode-hook`, `ess-mode-hook` and `ess-transcript-mode-hook`. Your systems administrator may have done this for you in '`ess-site.el`' (see Appendix A [Customization], page 64).

The font-lock patterns are defined in three variables, which you may modify if desired:

inferior-ess-font-lock-keywords Variable

Font-lock patterns for Inferior ESS Mode. The default value highlights prompts, inputs, assignments, output messages, vector and matrix labels, and literals such as '`NA`' and `TRUE`.

ess-mode-font-lock-keywords Variable

Font-lock patterns for ESS programming mode. The default value highlights function names, literals, assignments, source functions and reserved words.

ess-trans-font-lock-keywords Variable

Font-lock patterns for ESS Transcript Mode. The default value highlights the same patterns as in Inferior ESS Mode.

12.2 Parenthesis matching

Emacs and XEmacs have 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. Depending on what version of emacs you use, one of the following should work in your initialisation file:

```
(paren-set-mode 'paren) ;for XEmacs
(show-paren-mode t) ;for Emacs
```

12.3 Using graphics with ESS

One of the main features of the `S` package is its ability to generate high-resolution graphics plots, and ESS provides a number of features for dealing with such plots.

12.3.1 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.

12.3.2 Using ESS with windowing devices

Of course, the ideal way to use graphics with ESS is to use a windowing system. Under X11, this requires that the DISPLAY environment variable be appropriately set, which may not always be the case within your Emacs process. ESS provides a facility for setting the value of DISPLAY before the ESS process is started if the variable `ess-ask-about-display` is non-`nil`.

12.3.3 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.)

12.4 Imenu

Imenu is an Emacs tool for providing mode-specific buffer indexes. In some of the ESS editing modes (currently SAS and S), support for Imenu is provided. For example, in S mode buffers, the menubar should display an item called "Imenu-S". 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, e.g. 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`.

12.5 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 64, for ways to change the toolbar.

12.6 TAGS

The Emacs tags facility can be used to navigate around your files containing definitions of S functions. This facility is independent of ESS usage, but is written here since ESS users may wish to take advantage of TAGS facility. Read more about emacs tags in an emacs manual.

Etags, the program that generates the TAGS file, does not yet know the syntax to recognise function definitions in S files. Hence, you will need to provide a regexp that matches your function definitions. Here is an example call (broken over two lines; type as one line) that should be appropriate.

```
etags --language=none
--regex='/\([^ \t]+\)[ \t]*<-[ \t]*function/\1/' *.R
```

This will find entries in your source file of the form:

```
some.name <- function
```

with the function name starting in column 0. Windows users may need to change the single quotes to double quotes.

12.7 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.

To get started, first make sure you can load ess-rdired. Add the following to your .emacs and then restart emacs.

```
(autoload 'ess-rdired "ess-rdired"
  "View *R* objects in a dired-like buffer." t)
```

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).

13 Bugs and Bug Reporting, Mailing Lists

13.1 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.

13.2 Reporting Bugs

Please send bug reports, suggestions etc. to ESS-bugs@stat.math.ethz.ch

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.

13.3 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@stat.math.ethz.ch; contributions to the list may be mailed to ess-help@stat.math.ethz.ch. 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.

13.4 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 xemacs-www-faq* or *M-x view-emacs-FAQ*) and EmacsWiki (http://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`' 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.

Key (Character) Index

(Index is nonexistent)

Concept Index

O

P

Q

R

S

T

U

W

X

Variable and command index

Table of Contents