ESS — Emacs Speaks Statistics

(Documentation $Revision: 5.40 $ $Date: 2002/10/21 18:25:02 $)

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

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`, `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?

S is a powerful system for manipulating and analyzing data, but its user interface — particularly on Unix platforms — leaves something to be desired. ESS is a package which is designed to make **S** easier to use.

ESS provides several features which make it easier to interact with the **ESS** process (i.e. enter commands and view the output). 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 6.1 [Command-line editing], page 33.

- **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 6.5 [Command History], page 37.
- **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 6.2 [Completion], page 33.
- **Hot-keys** for quick entry of commonly-used commands in 'S' such as `objects()` and `search()`. See Section 6.7 [Hot keys], page 39.
- **Transcript recording** for a complete record of all the actions in an S session. See Section 6.4 [Transcript], page 35.
- **Interface to the help system**, with a specialized mode for viewing S help files. See Chapter 8 [Help], page 49.

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 42.
- **A specialized editing mode** for **S** code, which provides syntactic indentation and highlighting. See Section 7.5 [Indenting], page 44.
- **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 43.
- **Source code revision maintenance**, which allows you to keep historic versions of **S** source files. See Section 7.7 [Source Files], page 46.
- **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 43.

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

- **Evaluation of previously entered commands**, stripping away unnecessary prompts. See Section 6.4.3 [Transcript resubmit], page 36.

1.2 New features in ESS

Changes/New Features in 5.2.0:

- ESS[BUGS]: now supports interactive processing; special thanks to Aki Vehtari; new info documentation!
- ESS[SAS]: convert .log to .sas with ess-sas-transcript; THEN statements now highlighted correctly; info documentation improved; Local Variables bug fixes

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 GSASFILE 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 over 10 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

1.4 Getting the latest version of ESS

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

The latest public version of ESS can also be retrieved via cvs client. If you have a firewall, then you may need to take appropriate action.

The repository name is

```
:pserver:anoncvs@software.biostat.washington.edu:/var/anoncvs
```

Using a command-line cvs client (i.e. on Unix or DOS), first type:

```
cvs -d :pserver:anoncvs@software.biostat.washington.edu:/var/anoncvs login
```

The password is "anoncvs".

Then type:

```
cvs -d :pserver:anoncvs@software.biostat.washington.edu:/var/anoncvs co ess
```

1.5 How to read this manual

If you need to install ESS, read Appendix A [Installation], page 8 for details on what needs to be done before proceeding to the next chapter.

Appendix B [Customization], page 55 provides details of 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.

Don't forget that this manual is not the only source of information about ESS. In particular, the mode-based online help (obtained by pressing `C-h m` when in the process buffer, edit buffer or help buffer) is quite useful. However the best source of information is, as always, experience — try it out!

Appendix A 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 A.1 [Unix installation], page 8.

For Microsoft Windows Installation please skip to the See Section A.2 [Microsoft Windows installation], page 9.

A.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 gzipped tar file to ESSDIR.
3. Decompress/unarchive the files from the disribution.

   ```
   gunzip ess-VERSION.tar.gz
   tar xvf ess-VERSION.tar
   ```

   (or: `gunzip < ess-VERSION.tar.gz | tar xvf -`).
   (or using GNU tar: `tar zxvf ess-VERSION.tar.gz`).

   The `tar` command will create the subdirectory ess-VERSION and unarchive the files there.

   If you are using GNU Emacs 19.29, decompress/unarchive '`ESSDIR/ess-VERSION/lisp/19.29.tar.gz`', read '`ESSDIR/ess-VERSION/lisp/19.29/README`', follow the instructions and you might be able to get ESS to work. *Please note that GNU Emacs 19.29 is no longer supported*. For a list of supported versions of emacs, see See Section A.3 [Requirements], page 11.
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 Lisp path, you can do:

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

   to configure emacs for ESS.
6. That's it! 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).
7. (OPTIONAL) If you are running S-PLUS or R, you might consider installing the database files. From within emacs, `C-x d` to the directory containing ESS. Now:

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

   get running. once you have reached the SPLUS prompt, do:

   ```
   M-x ess-create-object-name-db
   ```

(this will create the file '`ess-s+6-namedb.el`'; if it isn't in the ESS directory, move it there).

Then, completions will be autoloaded and will not be regenerated for every session.

For R, do the same, using

```
M-x R
```

and then `M-x ess-create-object-name-db` creating '`ess-r-namedb.el`'; if it isn't in the ESS directory, move it there).

8. **(OPTIONAL) READ THIS ITEM THOROUGHLY BEFORE STARTING**:

   In the ESSDIR/ess-VERSION directory, edit the file '`Makeconf`' if you want to place the compiled files in other locations; see LISPDIR and INFODIR.

   Then type:

   ```
   make all
   ```

   If this works, then you might try:

   ```
   make install
   ```

   This will install the info files (and the lisp files, if they are to go in another directory). Don't forget to edit the file '`dir`' in the info directory specified by `INFODIR` in '`doc/Makefile`'. See the sample '`dir`' file for an example of the line to add.

   If you are using XEmacs, you might do:

   ```
   make EMACS=xemacs all
   ```

   and then

   ```
   make EMACS=xemacs install
   ```

   instead of editing the Makefile.

   Note that you might need to use **GNU make** for everything to work properly

   An alternative, if you are running XEmacs and have access to the XEmacs system directories, would be to place the directory in the site-lisp directory, and simply type `make all` (and copy the documentation as appropriate).

   For GNU Emacs, you would still have to move the files into the top level site-lisp directory.

A.2 Microsoft Windows installation

For **Microsoft Windows installation**, please follow the next steps: (see separate instructions above for UNIX See Section A.1 [Unix installation], page 8.

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". It will contain, at the end, either the tar file '`ess-VERSION.tar.gz`' or the zip file '`ess-VERSION.zip`', and a directory for the ESS source, which will be termed "the ESS-VERSION source directory".
2. Retrieve the compressed tar file '`ess-VERSION.tar.gz`' or the zipped file '`ess-VERSION.zip`' from one of the FTP or WWW archive sites via FTP (or HTTP). 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. Copy '`ess-VERSION.tar.gz`' to the location where you want the ess-VERSION directory, for example to '`c:\emacs\ess-VERSION.tar.gz`', and cd there. For example,

   ```
   cd c:\emacs
   ```

   Extract the files from the distribution, which will unpack into a subdirectory, '`ess-VERSION`'.

   ```
   gunzip ess-VERSION.tar.gz
   tar xvf ess-VERSION.tar
   (or: gunzip < ess-VERSION.tar.gz | tar xvf - ).
   (or: from the zip file: unzip ess-VERSION.zip)
   ```

   The `tar` command will extract files into the current directory.

   Do not create '`ess-VERSION`' yourself, or you will get an extra level of depth to your directory structure.
4. Windows users will usually be able to use the 'lisp/ess-site.el' as distributed. Only rarely will changes be needed.
5. 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\spls2000\cmd
   ```

   On Windows NT/2000, add the directories to the PATH using the MyComputer 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 "`\`".
6. Add the line

   ```
   (load "/PATH/ess-site")
   ```

   to your .emacs (or _emacs) file (or default.el or site-init.el, for a site-wide installation). Replace `/PATH` above with the value of ess-lisp-directory as defined in '`ess-site.el`'. Use forwardslashes `/`. (GNU Emacs uses the filename '`%HOME%/.emacs`' and XEmacs uses the filename '`%HOME%/.xemacs/init.el`' for the initialization file.)
7. 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).
8. To run statistical processes under emacs:

   Run S-PLUS 6.x or 2000 with:

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

   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. Similarly for S-PLUS 6.x. 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.

   Run S-PLUS 6.x or 2000 inside an emacs buffer

   ```
   M-x Sqpe+6
   ```

   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.

Similarly for S-PLUS 6.x. 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. You do not have access to interactive graphics in this mode. You get Unix-like behavior, in particular the entire transcript is available for emacs-style search commands.

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.

SAS for Windows uses the batch access with function keys that is described in '`doc/README.SAS`'. The user can also edit SAS files in an `ESS[SAS]` buffer and than manually copy and paste them into an Editor window in the SAS Display Manager.

For Windows, inferior SAS in an `iESS[SAS]` buffer does not work on the local machine. It does work over a network connection to SAS running on a remote Unix computer.

Reason: we use ddeclient to interface with programs and SAS doesn't provide the corresponding ddeserver capability.

9. (OPTIONAL) If you are running Sqpe or R, you might consider installing the database files. From within emacs, `C-x d` to the directory containing ESS. Now:

   ```
   M-x Sqpe+6
   ```

   (get running. once you have reached the SPLUS prompt, do:)

   ```
   M-x ess-create-object-name-db
   ```

   (this will create the file '`ess-s+6-namedb.el`'; if it isn't in the ESS directory, move it there).

   Then, completions will be autoloaded and will not be regenerated for every session.

   For R, do the same, using

   ```
   M-x R
   ```

   and then `M-x ess-create-object-name-db` creating '`ess-r-namedb.el`'; if it isn't in the ESS directory, move it there).
10. That's it!

A.3 Requirements

ESS works best with either GNU Emacs version 20.3 or higher, or XEmacs version 20.0 or higher. It has been most thoroughly tested with:

- S-PLUS versions 3.3, 3.4, 4.5, 5.0, 5.1, 6.0, 6.1
- R versions >=0.49
- S4
- SAS 6.x, 7.x, 8.x
- BUGS 0.5, 0.603
- Stata >=6.0

- XLispStat versions >=3.50

on the following platforms

- Solaris/SunOS (all)
- SGI (all)
- Linux (S4, S-PLUS 5.x, R, XLispStat, Stata 6.0)
- Microsoft Windows 95/98/NT/2000 (SPLUS 4.5 and 2000)
- Apple Mac OS (SAS for OS 9 and X11 R for OS X)

with the following versions of emacs

- GNU Emacs 20.3, 20.4, 20.5, 20.6, 20.7, 21.1, 21.3
- XEmacs 20.0, 20.4, 21.0, 21.1.13, 21.1.14, 21.4
- XEmacs 19.14, 19.16 and GNU Emacs 19.28, 19.29, 19.34[1]

A.4 Other variables you may need to change

If you run the **S** program (from the shell) with a command other than '`Splus`' you will need to set the variable `inferior-ess-program` to the name of the appropriate program by including a line such as

```
(setq inferior-ess-program "S+")
```

in your '`.emacs`' file (substituting '`S+`' for the name of your **S** program.)

If you need to call this program with any arguments, the variable you need to set is dependent on the value of `inferior-ess-program`; for example if it is `"Splus"`, set the variable `inferior-Splus-args` to a string of arguments to the `Splus` program. If `inferior-ess-program` has some other value, substitute the `Splus` part of `inferior-Splus-args` with the appropriate program name. There aren't many instances where you need to call **S** with arguments, however: in particular do not call the **S** program with the '`-e`' command-line editor argument since ESS provides this feature for you.

If you are running Splus (the enhanced version of **S** from Statsci) you may also need to set the variable `S-plus` to `t`. If your value of `inferior-ess-program` is `"S+"` or `Splus` this will not be necessary, however; `S-plus` defaults to `t` in this case.

Finally, if you use a non-standard prompt within S, you will need to set the variable `inferior-ess-prompt` to a regular expression which will match both the primary prompt (`"> "` by default) and the continuing prompt (default of `"+ "`.) The default value of this variable matches S's default prompts. For example, if you use (`"$ "`) as your primary prompt (you have `options(prompt="$ ")` in your `.First` function), add the following line to your '`.emacs`':

```
(setq inferior-ess-prompt "^\\(\\+\\|[^\\$]*\\$\\) *")
```

You will also need to set the variable `inferior-ess-primary-prompt` to a regular expression which matches the primary prompt only. Do not anchor the regexp to the beginning of the line with '`^`'. Once again, the default value matches S's default prompt; in the example above the appropriate value would be `"[^\\$]*\\$ *"`.

[1] Note that you must have '`custom`' support. It is available at The Custom Library. These releases of emacs are no longer supported, so an upgrade is recommended if you plan to use ESS. If you have GNU Emacs 19.29, See Section A.1 [Unix installation], page 8.

Once these variables are set appropriately, ESS should work on any system.

2 Starting the ESS process

To start an **S** session, simply type *`M-x S RET`*, i.e. press ⟨ESC⟩, then ⟨x⟩, then capital ⟨S⟩ and then the ⟨RETURN⟩ key.

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.

2.1 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 for the size of memory allocated to the R process, for example.

You can switch to any active **ESS** process with the command *`C-c C-k`* (`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.

For the predecessor to ESS (S-mode 4.8), the initial process was not visibly numbered, i.e. S instead of S1 was used in the mode-line. To obtain this behavior, set the variable `ess-plain-first-buffername` to `t`. See '`ess-site`' for how to set this for all users.

2.2 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.
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 and lets SAS display them formatted correctly as in a SAS log file.

4. Graphics (interactive) on the remote machine. If you run X11 (See Section 10.2.2 [X11], page 53, 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 X-windows 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 4.5 [ESS(SAS)–Function keys for batch processing], page 24, 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 2.3 [S+elsewhere and ess-elsewhere], page 15.

2.3 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 X-windows graphics on the remote unix machine you should be able to display them in '`xfree86`' on your PC. If you don't run X-windows, then you can write graphics to the postscript device and copy it to your PC with dired and display it with ghostscript.

2.4 Changing the startup actions

If you do not wish ESS to prompt for a starting directory when starting a new process, set the variable `ess-ask-for-ess-directory` to `nil`. In this case, the 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 records 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 6.4.4 [Saving transcripts], page 37.)

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.

3 Help for the S family

3.1 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.

3.2 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 R
```

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

M-x S

Note that R has some extremely useful command line arguments, -v and -n. To enter these, call R using a "prefix argument", by

C-u M-x R

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

-v 10000 -n 5000

Then that R process will be started up using "R -v 10000 -n 5000".

New for ESS 5.1.2 (and later): "S-elsewhere" command

The idea of "M-x S-elsewhere" is that we open a telnet (or rlogin) to another machine, call the buffer "*S-elsewhere*", and then run S on the other machine in that buffer. We do that by defining "sh" as the inferior-S-elsewhere-program-name. Emacs sets it up in a "*S-elsewhere*" iESS buffer. The user does a telnet or login from that buffer to the other machine and then starts S on the other machine. The usual C-c C-n commands from myfile.s on the local machine get sent through the buffer "*S-elsewhere*" to be executed by S on the other machine.

3.3 Handling and Reusing Transcripts

- edit transcript

- color and font highlighting based on syntax.

- resubmit multi-line commands to an active process buffer

- The ability to request help from an S process for variables and functions, and to have the results sent into a separate buffer.

- ability to switch between processes which would be the target of the buffer (for the above).

3.4 ESS-help–assistance with viewing help

- move between help sections
- send examples to S for evaluation

3.5 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 (GNU emacs uses the filename '`~/.emacs`' and Xemacs uses the filename '`~/.xemacs/init.el`' for the initialization 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.

3.6 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")
```

3.7 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-s+3.el and essd-s4.el files to be "S" instead of "S4" and "S+3". This will insure 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.

4 Help 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. By emacs, we mean either GNU Emacs from the Free Software Foundation or XEmacs from the XEmacs Project.

4.1 ESS[SAS]–Design philosophy

ESS[SAS] was designed to aid the user in writing and maintaining SAS programs, such as myfile.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.

4.2 ESS[SAS]–Editing files

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

- proper indenting, generated by both [Tab] and [Return].
- color and font choices based on syntax.
- 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).
- 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 automatically refreshed with each keypress).

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 4.3 [iESS(SAS)–Interactive SAS processes], page 23.)

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 a DO ... TO or a DO ... TO ... BY statement, TOs are not highlighted (and neither is BY).

4.3 iESS[SAS]–Interactive SAS processes

iESS (inferior ESS) 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 following settings for SAS command-line options (the default of inferior-SAS-args):

```
-stdio -linesize 80 -noovp -nosyntaxcheck
```

-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 myfile.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 myfile.lst in SAS batch usage and to the "SAS: OUTPUT" window in the SAS Display Manager. All data related printed output from the PROCs appear in this window.

The iESS [SAS:1] buffer exists solely as a communications buffer. Files are edited in the myfile.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 the 'Local' 'Submit' menu commands. The user should never use this buffer directly.

Troubleshooting: See Section 4.8 [iESS(SAS)–Common problems], page 29.

4.4 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 AppleScript is used ('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. 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 ess-kermit-get which brings the contents of the remote file into your local copy. Also, 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, you can set it in Local Variables: at the end of your program). 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 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 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 ess-sas-submit-command "c:/progra~1/sas/sas.exe")
;Windows example
(setq 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 file by:

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

4.5 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 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, say 2, you must turn it off before selecting a different option, say 3.

   ```
   (load "ess-site")
   (setq ess-sas-local-unix-keys nil) ;;2
   (setq ess-sas-local-pc-keys t)     ;;3
   ```

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	Description
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 FSEDIT/FSBROWSE session on the SAS dataset near point

F10 F10 toggle-log
: toggle ESS[SAS] for .log files; may be 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-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

SUBMIT, PROGRAM, LOG and OUTPUT need no further explanation since they mimic the SAS Display Manager 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 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 LIBNAMEs defined in autoexec.sas, then the defaults for these variables should be sufficient.

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 point 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 two variables for your own situation:

```
(setq ess-sas-graph-suffix-regexp "[.]\\(e?ps\\|gif\\|jpe?g\\|tiff?\\)")
(setq ess-sas-image-viewer "kodakimg") ;; Windows default
```

4.6 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 4 spaces from the margin, continuation lines are indented 4 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 0)
```

ESS provides an alternate behavior for the TAB key that makes it behave as it does in SAS Display Manager, i.e. move the cursor to the next tab stop. The alternate behavior also provides a backwards TAB, C-TAB, that moves the cursor to the tab 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 1)
```

Under the alternate behavior, the TAB key is bound to tab-to-tab-stop and the tab stops are set at multiples of sas-indent-width.

4.7 ESS[SAS]–Usage scenarios

We present a batch and an interactive scenario for using ESS with SAS. The remarks with respect to graphics apply to either with exceptions noted.

Graphics

Output from GPROCs can be displayed in a SAS/Graph window for SAS batch on Windows or for both SAS batch and interactive with X11 on Unix. If you need to create graphics files and view them with F12, then include the following in myfile.sas (for F12 to work the FILENAME statement must be in myfile.sas, but the GOPTIONS statement can be in your 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;
```

SAS Batch (ess-sas-global-unix-keys keys shown, ess-sas-global-pc-keys in parentheses).

Open the file you want to work with.

```
C-x C-f myfile.sas
```

myfile.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 myfile.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 %1
```

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 1st 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)
```

Interactive SAS

Open the file you want to work with.

```
C-x C-f myfile.sas
```

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

```
M-x SAS
```

Four buffers will appear on screen:

```
Buffer          Mode            Description
myfile.sas      ESS[SAS]        your source file
*SAS:1*         iESS [SAS:1]    ESS 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 myfile.sas. Enter the sequence:

```
C-c C-w
```

The cursor will normally be in buffer myfile.sas. If not, put it there:

```
C-x b myfile.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 the keyword '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 myfile.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 myfile.log and myfile.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 myfile.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!

4.8 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 (in the ESS source directory) needs to be
     executable (solution: "chmod ugo+rx ess-sas-sh-command").
   - sas isn't in your executable path (verify using "which sas" from
     a shell command-line)

3. M-x SAS starts SAS Display Manager.  Probably, the command "sas"
   on your system calls a shell script.  Specify the path to the real "sas"
   executable in the file ess-sas-sh-command, i.e.:
      /usr/local/sas612/sas </dev/tty 1>$stdout 2>$stderr $@
   To find the "sas" exectuable, you can execute the unix command:
      find / -name sas -print
```

4.9 ESS[SAS]–MS Windows

- iESS[SAS] does not work on Windows. See Section 4.8 [iESS(SAS)–Common problems], page 29.

- ESS[SAS] mode for editing SAS language files works very well. See Section 4.2 [ESS(SAS)–Editing files], page 22.
- There are two execution options for SAS on Windows. You can use batch. See Section 4.4 [ESS(SAS)–Batch SAS processes], page 23.

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

5 Help 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.

5.1 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 4.5 [ESS(SAS)–Function keys for batch processing], page 24. 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.

5.2 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.

5.3 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.

6 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.

6.1 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.

6.2 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-name-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 6.6 [History expansion], page 38.

6.3 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. On some systems, however, this can be rather slow; it's doubly frustrating when you consider that most of the directories on the search list are the standard **S** libraries, which never change anyway! When ESS was installed, a database of the standard object names should have been created which should speed up this process at the start of an S session; if it has not been created you will get a warning like 'S-namedb.el does not exist'. See Appendix A [Installation], page 8, for information on how to create this database.

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

[1] The variable `ess-change-sp-regex` 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.

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.

6.4 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*.

6.4.1 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`; for information on this and other options for handling process input and output see section "Shell Mode Options" in *The GNU Emacs Reference Manual*.)

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.

6.4.2 Viewing more historic 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* [Command]
comint-forward-matching-input *regexp arg* [Command]

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.

6.4.3 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 signaled). 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 signaled. 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 6.5 [Command History], page 37.

6.4.4 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 2.4 [Customizing startup], page 16. We recommend you save your transcripts with filenames that end in '`.St`'. There is a special mode (ESS transcript mode — see Chapter 9 [Transcript Mode], page 51) 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 it is your intention 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-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!

6.5 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.

6.6 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`*.

6.7 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 `es-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 43, 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 8 [Help], page 49 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.

6.8 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.

6.9 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 42.

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

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-insert-function-templates` is non-`nil`, you will be presented with a template defined by `ess-function-template` 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 46) 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-x ‘`* (`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-x ‘`* 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-x ‘`*.

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`* (`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-next-line`)
  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-e (`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`.

*** Maybe a link to customization section here ***

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. See Section B.1.3 [Indentation variables], page 56.

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. (GNU emacs uses the filename '`~/.emacs`' and Xemacs uses the filename '`~/.xemacs/init.el`' for the initialization 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).
- *`LFD`* (`newline-and-indent`)
  Insert a newline, and indent the next line. (Note: if your keyboard does not have a ⟨LINEFEED⟩ key, you can use *`C-j`* instead.) Some people prefer to bind ⟨RET⟩ to this command.
- *`ESC C-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 a comment line appropriately, or inserts an empty (single-'`#`') comment.
- *`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 see Section B.1.3 [Indentation variables], page 56 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-e`* (`ess-beginning-of-function`)
  Moves point to the beginning of the function containing point.
- *`ESC C-a`* (`ess-end-of-function`)
  Moves point to the end of the function containing point.
- *`ESC C-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 is.

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 6.9 [Other], page 41, 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 really 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) been 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 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 Reading help files in ESS

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:

- *`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 42). 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.

9 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 6 [Entering commands], page 33): 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`.

9.1 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.

9.2 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 43) using the code evaluation commands from **S** mode, also available in **S** Transcript Mode.

10 Other features of ESS

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

10.1 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 the 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` (see Section B.2 [Hooks], page 58). Your systems administrator may have done this for you in '`ess-site.el`' (see Appendix B [Customization], page 55).

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

ess-inf-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-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 stuff as in Inferior **ESS** Mode.

10.2 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.

10.2.1 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.

10.2.2 Using ESS with windowing devices

Of course, the ideal way to use graphics with ESS is to use a windowing system. Under X windows, 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`. See Appendix B [Customization], page 55, for details of this variable, and see Chapter 2 [Starting up], page 14 for information on how to set the value of DISPLAY when beginning an **S** session.

10.3 Object Completion

If you are running S-PLUS or R, you might consider installing the database files. From within (X)Emacs, "C-x d" to the directory containing ESS. Now:

`M-x S+3`

(get S-PLUS running. once you have reached the SPLUS 3.x prompt, do:)

`M-x ess-create-object-name-db`

(this will create the file: ess-s+3-namedb.el; if it isn't in the ESS directory, move it there).

Then, completions will be autoloaded and will not be regenerated for every session.

For R:

`M-x R`

(get R running. once you have reached the R prompt, do:)

`M-x ess-create-object-name-db`

(this will create the file: ess-r-namedb.el; if it isn't in the ESS directory, move it there).

11 Bugs and Bug Reporting, Mailing Lists

11.1 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.

11.2 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.

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

11.3 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.

Appendix B 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.

B.1 Variables for customization

ESS is easily customizable by means of setting variables in your '`.emacs`' file. In most cases, you can change defaults by including lines of the form

`(setq` *variable-name value*`)`

in your '`.emacs`'.

In what follows, variable names will be listed along with their descriptions and default values. Just substitute the variable name and the new value into the template above.

B.1.1 Variables for starting ESS

ess-ask-for-ess-directory [User Option]

Default: `t`

If this variable has a non-`nil` value, then every time ESS is run with *`M-x S`* you will be prompted for a directory to use as the working directory for your **S** session; this directory should have a '`.Data`' subdirectory. If the value of `ess-ask-for-ess-directory` is `nil`, the value of `S-directory` is used as the working directory.

S-directory [User Option]

Default: Your home directory

The working directory for your **S** session if `ess-ask-for-ess-directory` is `nil`, and the default when prompting for a directory if it is not. For example, you may wish to set this to the value of the current buffer's working directory before starting **S** by adding the following line to your '`.emacs`' file (see Section B.2 [Hooks], page 58)

```
(setq ess-pre-run-hook
   '((lambda () (setq S-directory default-directory))))
```

ess-ask-about-display [User Option]

Default: `nil`

If this variable has a non-`nil` value, then every time ESS is run with *`M-x S`* you will be asked for a value for the `DISPLAY` environment variable to be used in the current **S** session. If this variable is not set correctly, **S** will not be able to create any windows under the X windowing environment. Completion is provided over the `X-displays-list` variable; the default is the current value of `DISPLAY`. This feature is useful is you often run **S** on a different display than that of the machine you are running **S** from. If `ess-ask-about-display` is `nil`, the current value of `DISPLAY` is used.

X-displays-list [User Option]
Default: `'(":0.0")`
List of possible values for the `DISPLAY` environment variable, provided for completion when prompting for such a value.

B.1.2 Variables for dump files

ess-insert-function-templates [User Option]
Default: `t`
If this variable has a non-`nil` value, then dumping a non-existent object will result in the edit buffer containing a skeleton function definition, ready for editing.

ess-source-directory [User Option]
Default: `"/tmp/"`
Directory name (ending in '`/`') in which dump files are placed. This should always be a writable directory.

ess-source-directory-generator [User Option]
Default: `nil`
Alternative, dynamic method of specifying the directory for dump files.

ess-dump-filename-template [User Option]
Default: *user_name*`.`*object_name*`.S`
Naming system to use for dumped object files. See Section 7.8 [Source Directories], page 47, for details of this and the previous two variables.

ess-keep-dump-files [User Option]
Default: `nil`
Boolean flag signifying whether to keep dump files or to delete them after each use. See Section 7.7 [Source Files], page 46, for more details.

B.1.3 Variables controlling indentation

ess-tab-always-indent [User Option]
Default: `t`
If non-`nil`, then *`TAB`* in the edit buffer always indents the current line, regardless of the position of point in the line. Otherwise, indentation is only performed if point is in the lines indentation, and a tab character is inserted is point is after the first nonblank character.

ess-auto-newline [User Option]
Default: `nil`
Non-`nil` means automatically newline before and after braces inserted in **S** code.

The following variables control amounts of indentation. These variables automatically become buffer-local in any ESS buffer, and so setting any of these variables has effect in the current buffer only.

ess-indent-level [User Option]
Default: 2
Extra indentation of **S** statement sub-block with respect to enclosing braces.

ess-brace-imaginary-offset [User Option]
Default: 0
Extra indentation (over sub-block indentation) for block following an open brace which follows on the same line as a statement.

ess-brace-offset [User Option]
Default: 0
Extra indentation for braces, compared with other text in same context.

ess-continued-statement-offset [User Option]
Default: 0
Extra indent for lines not starting new statements.

ess-continued-brace-offset [User Option]
Default: 0
Extra indent for substatements that start with open-braces. This is in addition to `ess-continued-statement-offset`.

ess-arg-function-offset [User Option]
Default: 2
Extra indent for arguments of function `foo` when it is called as the value of an argument to another function in `arg=foo(...)` form. If not number, the statements are indented at open-parenthesis following `foo`.

ess-expression-offset [User Option]
Default: 4
Extra indent for internal substatements of the call to `expression()` specified in

```
obj <- expression(...)
```

form. If not a number, the statements are indented at open-parenthesis following '`expression`'.

ess-else-offset [User Option]
Default: 2
Extra indentation of the `else` clause with respect to the corresponding `if`.

In addition, a number of default styles are defined for you (in `ess-style-alist`):

ess-default-style [User Option]

Default: `DEFAULT`

The default formatting style to use in edit buffers. The DEFAULT style uses the values of the above indentation variables at load-time, so that changing these variables in your '`.emacs`' file will affect your buffer defaults. Other styles are: `GNU`, `BSD`, `K&R`, `CLB`, and `C++`. See '`ess-cust.el`' for the complete definitions of the styles. See Section 7.1 [Edit buffer], page 42, for more details.

B.1.4 Variables controlling interaction with the ESS process

comint-input-ring-size [User Option]

Default: 50

Number of commands to store in the command history.

ess-execute-in-process-buffer [User Option]

Default: `nil`

If this is `nil`, then the `ess-execute-` commands (see Section 6.9 [Other], page 41) output to a temporary buffer. Otherwise, the output goes to the **ESS** process.

ess-eval-visibly-p [User Option]

Default: `nil`

If non-`nil`, then the `ess-eval-` commands (see Section 7.1 [Edit buffer], page 42) echo the **S** commands in the process buffer by default. In any case, passing a prefix argument to the eval command reverses the meaning of this variable.

B.2 Customizing ESS with hooks

ESS provides the following hooks:

ess-mode-hook [Hook]

Called every time `ESS` is run.

ess-mode-load-hook [Hook]

Called just after the file '`ess.el`' is loaded. Useful for setting up your keybindings, etc.

ess-pre-run-hook [Hook]

Called before the **ESS** process is started (e.g., with *`M-x S`*). Good for setting up your directory.

ess-post-run-hook [Hook]

Called just after the **ESS** process starts up. This can be used to evaluate **ESS** code at the start of a session, with `ess-eval-visibly`, say.

inferior-ess-mode-hook [Hook]

For customizing inferior ess mode. Called after inferior-ess-mode is entered and variables have been initialized.

ess-help-mode-hook [Hook]

Called every time when entering ess-help-mode (i.e., an **ESS** help buffer is generated).

ess-send-input-hook [Hook]

Called just before line input is sent to the process.

ess-transcript-mode-hook [Hook]

For customizing ESS transcript mode.

B.3 Changing the default ESS keybindings

ESS provides a separate keymaps (keymap variables) for **ESS** process buffers, edit buffers and for help buffers. The key bindings in the edit buffers further depend on the language and dialect in use.

inferior-ess-mode-map [Keymap]

Keymap used in the **ESS** process buffer. The bindings from `comint-mode-map` are automatically inherited.

ess-mode-map [Keymap]

Keymap used within edit (ess-mode) buffers.

ess-eval-map [Keymap]

Keymap used within edit buffers for sending ESS code to the running process.

ess-help-mode-map [Keymap]

Keymap used within help buffers. In addition, `ess-help-sec-map` is the keymap for the '`s`' prefix key. Keys defined in `ess-help-sec-keys-alist` are automatically inserted into this keymap when ESS is loaded.

ess-transcript-mode-map [Keymap]

Keymap used within ESS transcript buffers.

Key (Character) Index

(Index is nonexistent)

Command and Function Index

Concept Index

Variable and command index

Table of Contents