RJE Python Appendices

Richard J. Edwards © 2009.

Contents

Tables

1. Introduction

This manual is designed to accompany the manuals of specific RJE python programs and gives an overview of general commandline options etc. that are common across applications.

The manual was last edited on 22 November 2012.

1.1. Using this Manual

As much as possible, I shall try to make a clear distinction between explanatory text (this) and text to be typed at the command-prompt etc. Command prompt text will be `written in Courier New` to make the distinction clearer. Program options, also called 'command-line parameters', will be **`written in bold Courier New`** (and coloured **`red`** for fixed portions or **`dark`** for user-defined portions, such as file names etc.). Optional parameters will (if I remember) be `[`in square brackets`]`. Names of files will be marked in **coloured normal text**.

1.2. Getting Help

Much of the information here is also contained in the documentation of the Python modules themselves. A full list of command-line parameters can be printed to screen using the **`help`** option, with short descriptions for each one:

```
python program.py help
```

Details of command-line options specific to each program can be found in the distributed **readme.txt** and **readme.html** files.

If still stuck, then please e-mail me (seqsuite@gmail.com) whatever question you have. If it is the results of an error message, then please send me that and the log file (see Chapter 2) too.

1.3. Availability and Local Installation

These programs should work on any system with Python installed without any extra setup required. If you do not have Python, you can download it free from www.python.org at http://www.python.org/download/. The modules are written in Python 2.x and most have been tested with 2.7. The Python website has good information about how to download and install Python but if you have any problems, please get in touch and I will help if I can.

All the required files should have been provided in the download zip file. The Python Modules are open source and may be changed if desired, although please give me credit for any useful bits you pillage. I cannot accept any responsibility if you make changes and the program stops working, however!

Note that the organisation of the modules and the complexity of some of the classes is due to the fact that most of them are designed to be used in a number of different tools. As a result, not all the options listed in the __doc__() (**`help`**) will be of relevance. If you want some help understanding the way the modules and classes are set up so you can edit them, just contact me.

2. Fundamentals

2.1. Installation and setup

All the RJE_Python programs and documentation are now released as three packages: SLiMSuite, SeqSuite and RJESuite. These all have the same basic installation, directory structure and setup requirements. For basic functionality, no other setup should be necessary beyond downloading and unzipping the package in the desired directory if Python is installed on your system (see 1.3). Some programs will need to use external components or accessory applications (see 2.3), which may need additional installation.

2.1.1. Directory Structure

Once unzipped, RJESuite will unpack the following directories:

- **data/**. This contains example data for testing programs. (Note that this directory may be absent.)
- **docs/**. This contains the documentation (such as this manual)
- **extras/**. This contains accessory programs that are not part of the main program suite.
- **libraries/**. This contains all the python libraries used by the main program suite (and extras).
- **settings/**. This is where the default INI files (see 2.2.4) are placed to set default settings.
- **tools/**. This contains the main program suite.

2.2. Command-line Options

2.2.1. How to Use this Section

This section lists the general Command-line options that are used in multiple programs and may not be listed in their individual documentation. These form part of the parent RJE_Object class inherited by all RJESuite classes. Default values are given [in square brackets]. This information is also available by printing the __doc__ attribute of the **rje.py** module at a Python prompt, or using the `help` option:

`print rje.__doc__` (in Python)

`python rje.py help` (commandline)

Please contact me if you want any further details of a specific option and/or advice as to when (not) to use it.

2.2.2. Option Types

There are essentially three types of command-line option:

1. Those that require a value (numerical or text), `option=X`. Those that require a filename as the value will be witten: `option=FILE`. Those that require a directory path as the value will be witten: `option=PATH`. Those that lead to an accessory application (rather than just its path) may also be listed as `option=COMMAND`. Paths and filenames should always use forward slash (/) separators, whatever the operating system.
2. True/False (On/Off) options, `option=T/F`. For these options:
    a. `option=F` and `option=False` are the same and turn the option off.
    b. `option`, `option=T` and `option=True` are the same and turn the option on.
3. List options. These are like the value options but have multiple values, separated by commas: `option=X,Y`. Where `..` is used, the number elements is optional, e.g. `option=X,Y,..,Z` could take `option=X` or `option=A,B,C,D`. Where `option=LIST` is used, the number of elements is optional and `LIST` could actually be the name of a file containing the list of elements.

As well as giving options in the `option=VALUE` format, commands can also be provided in a more traditional UNIX format: `-option VALUE`. The two formats can be freely mixed:

```
python rje.py -v 1 i=0 -newlog
```

2.2.3. Long option values, whitespace and special characters

Some characters, such as whitespace, commas, pipes ("|") and ampersands, will be interpreted by UNIX in particular ways from the commandline. If you have such characters within the option value, then either place the settings in an INI file (2.2.4) or enclose the option value in quotes. If the value contains whitespace, double quotes will be needed even with the INI file, as whitespace is used to delimit commandline options. *E.g.*

```
python rje.py option="Two words" limits="2,3"
```

2.2.4. INI Files

As well as feeding commands in on the command-line, any options listed can also be save in a plain text file and called using the option `ini=FILE`. Automatically, the program will read in any options from the file named after the program (e.g. **haqesac.ini**) and **rje.ini**, if present. Any text on a line following a hash ("#") will be treated as a comment and ignored unless it is part of an option value in double quotes. This allows INI files to be documented.

The program will first look in the current (run) directory for these INI files. If not found, it will look in the settings/ directory of the SLiMSuite or SeqSuite installation. Universal default values for all runs can therefore be put in this settings/ directory.

It is recommended that an **rje.ini** file is made and placed in the settings/ directory. This file should contain the paths to the programs listed in **2.3 External Programs Used by RJE Programs**:

```
blastpath=PATH

fastapath=PATH

clustalw=COMMAND

muscle=COMMAND
```

Note that the first two are just paths to the programs, while for ClustalW and MUSCLE the actual program commands themselves must be included. This is to make it easier to replace these programs with alternatives. (See **3.3 Replacing Components with Other Programs**.)

If running in windows, it is also advisable to add the `win32=T` command to the ***.ini** file.

NB. For `PATH` variables, directories should be separated by a forward slash (`/`). If paths contain spaces, they should be enclosed in double quotes: `path="example path"`. It is recommended that paths do not contain spaces as function cannot be guaranteed if they do.

2.2.5. Option Precedence

Later options will supersede earlier ones if they are mutually exclusive. Options from an ini file will be inserted into the list at the point the ini file is called. Options from the **rje.ini** file are read first, followed by the **program.ini** file) This means that ini file options can be over-ruled, e.g.

`xxx.py ini=eg.ini i=1` will supersede any interactivity setting in **eg.ini** with `i=1`.

`xxx.py i=1 ini=eg.ini` will use any interactivity setting in **eg.ini** and over-rule `i=1`.

2.2.6. Interactivity and Verbosity settings

By default, the programs are generally setup to run through to completion without any user-interaction if given all the options it needs. For more interaction with the program as it runs, use the argument '`i=1`'.

```
python xxx.py commandlist i=1
```

Both the level of interactivity and the amount printed to screen can be altered, using the interactivity [`i=X`] and verbosity [`v=X`] command-line options, respectively, where `X` is the level from none (-1) to lots (2+). Although in theory `i=-1` and `v=-1` will ask for nothing and show nothing, there is a good chance that some print statements will have escaped in these early versions of the program. There is also the possibility that accessory programs may print things to the screen beyond the control of the calling program.

Please report any irritations and suggestions for changes to what is printed at different verbosity levels.

2.2.7. General Command-line Options

The most common command-line options used in RJE programs are listed in Table 1. The Module column gives some information on which programs/modules are affected by the command:

- all = all modules use this command
- most = most of the programs in RJESuite will use this command
- limited = only a few programs in RJESuite use this command

To be safe, you should refer to the specific program documentation, especially if the program behaves counter to expectation.

2.3. External Components of RJE Programs

In addition to the python modules listed above, some of the programs commonly make use of the following published programs. These are freely available for downloading and installing. It is recommended that the user downloads and installs these programs according to the instructions given on the appropriate website.

ALIGN: This is part of the Fasta package (Pearson, 1994; Pearson, 2000) and can be downloaded from the University of Virginia: http://fasta.bioch.virginia.edu/. Make sure that align is part of the download. For some reason it seems to have been dropped from later packages. You may need to install an earlier package first (e.g. 2.1) and then a later package.

BLAST: BLAST (Altschul, et al., 1990) is freely available for download from NCBI at: http://www.ncbi.nlm.nih.gov/blast/download.shtml.

CLUSTALW: ClustalW (Higgins and Sharp, 1988; Thompson, et al., 1994) is an old stalwart for bioinformatics and is freely available from EMBL: ftp://ftp-igbmc.u-strasbg.fr/pub/ClustalW/. Note that CLUSTALW is used as a backup for MUSCLE (below) and to draw trees. See **Replacing Components with Other Programs** for details of how to incorporate other tree-drawing packages.

MUSCLE: MUSCLE (Edgar, 2004) is a newer multiple alignment program available from http://www.drive5.com/muscle.

R: The statistical programming language, R, is used for PNG visualisation by some RJESuite programs. R is freely available from: http://cran.r-project.org/. Note that some installations of R can require a bit of tweaking of the R scripts provided (in **libraries/r/**). Please email seqsuite@gmail.com if you require some help with this and/or have problems with the R-coded PNG visualisations.

It is recommended that paths to these programs are placed into an **INI** file (see Error! Reference source not found. **Setting up the INI File**).

These can usually be replaced with different programs if desired (see **3.3 Replacing Components with Other Programs**).

2.4. Log Files

Every program generates a log file when it is run. By default, this file will be named after the calling program (e.g. **gasp.py** will produce a log called **gasp.log**) but this can be changed with the `log=FILE` option. Logs will be appended unless the `newlog` (or `newlog=T`) option is used.

The log file records information that may help subsequent interpretation of results or identify problems. Probably it's most useful content is any error messages generated, which are marked by a #ERR line header. Other information is also recorded along with the runtime (HH:MM:SS since the program started).

For help interpreting log files, please check the relevant software manual or contact me if the information is missing. (Hopefully, the log content is mostly self-explanatory but I shall add any explanations I have to send people to the relevant manual's appendix.)

Table 1. Common command-line options.

Option	Description	Default*	Module
	General Input/Output Options		
v=X	Sets verbosity (-1 for silent)	[0]	all
i=X	Sets interactivity (-1 for full auto)	[0]	all
log=FILE	Redirect log to FILE	[program.log]	all
newlog=T/F	If set to True will not write to screen *or* log.	[False]	all
silent=T/F	Create new log file.	[False]	all
errorlog=FILE	If given, will write errors to an additional error file.	[None]	all
help	Prints help documentation to screen.	[False]	all
basefile=FILE	This will set the 'root' filename for output files (FILE.*), including the log	[None]	limited
outfile=FILE	This will set the 'root' filename for output files (FILE.*), excluding the log.	[None]	limited
delimit=X	Sets standard delimiter for results output files.	[varies]	most
mysql=T/F	"MySQL output" with lowercase headers that lack spacers.	[False]	most
append=T/F	Append to results files rather than overwrite.	[False]	most
force=T/F	Force to regenerate data rather than keep old results.	[False]	limited
backups=T/F	Give option to backup certain files if append=F.	[True]	most
maxbin=X	Maximum number of trials for using binomial (else use Poisson)	[-]	limited
	System Info		
win32=T/F	Run in Win32 Mode	[False]	all
memsaver=T/F	Run in "Memory Saver" mode	[False]	limited
runpath=PATH	Run program from given path (log files and some programs only)	[PATH called from]	limited
rpath=PATH	Path to installation of R	['c:\\Program Files\\R\\R-2.6.2\\bin\\R.exe']	limited
	Forking		
forks=X	Number of forks	[0]	limited
killforks=X	Number of seconds of inactivity before killing forks	[3600]	limited
noforks=T/F	Option to over-ride and cancel forking	[False]	limited

*Please note: defaults might vary from program to program.

3. Appendices

3.1. Troubleshooting & FAQ

Currently, this is a small section as I have not had enough feedback to have FAQs, or anything like that. Here is a list of things that I think MAY cause problems to the unwary:

- Giving file names with spaces without enclosing in double quotes “”. (Only the first word will be taken as the filename.) It is not recommended to have spaces in filenames as some programs (and accessory programs) may go wrong if you do.
- Including spaces in paths to programs etc. without double quotes “”. Likewise, a lack of spaces altogether is favourable.
- Incorrect formatting of input files. Check instructions in relevant manual. If things are still not working, check that the line breaks are \n and not \r or some other odd format.
- I’m not sure when but there is a possibility of problems if running in Windows without the **`win32=T`** option.
- Using out-of-date modules. Sometimes changes to a program will actually be made in one of the modules they import: if upgrading, upgrade all modules and not just the program being called.

3.2. Abbreviations and Glossary

The following terms and abbreviations are used in this manual:

- **GO.** Gene Ontology. (See www.geneontology.org.)
- **PPI**. Protein-protein interaction.

3.3. Replacing Components with Other Programs

The most important functions performed by the external programs alignment and tree-drawing. This section lists some ways to incorporate alternative programs for these functions into RJE programs. I am always interested to add more functionality, so if there is a program you would like to use instead of those listed, then please contact me and I may be able to add them in a more controlled fashion than below.

3.3.1. Alignment programs

By default, MUSCLE (Edgar, 2004) is used for alignments as I have found this to be both fast and accurate. There can be problems with memory allocation for larger datasets and so and ClustalW (Higgins and Sharp, 1988; Thompson, et al., 1994) is used for large datasets above a certain total number of residues (as determined by the **`cwcut=X`** parameter). Either of these programs can be replaced, however, by another program that uses the same command-line format call the programs.

For MUSCLE, the system call is:

muscle **`-in INFILE -out OUTFILE`**, where **`INFILE`** and **`OUTFILE`** are both fasta format.

The path to MUSCLE can be changed to redirect to another program using the **`muscle=PATH`** option.

For ClustalW, the system call is:

clustalw **`INFILE`**, where **`INFILE`** is in fasta format (***.fas**) and the output file (***.aln**) is in ClustalW align format.

The path to ClustalW can be changed to redirect to another program using the **`clustalw=PATH`** option.

3.3.2. Tree-drawing programs

The default for RJESuite programs is to use the Neighbour-joining method implemented in ClustalW for drawing trees. Although this is not the most accurate phylogeny construction algorithm around, it is fast and efficient and reasonable for trees of closely-related sequences with high bootstrap support, such as those HAQESAC was designed to build and work with. Again, this program can be replaced with another using the **maketree=PATH** option. The system call used is:

clustalw **-infile=INFILE -bootstrap=X -seed=X [-kimura]** for UNIX, or

clustalw **INFILE -bootstrap=X -seed=X [-kimura]** for Windows, where **INFILE** is in fasta format (***.fas**) and the output file (***.phb**) is in bootstrapped Phylip format (I think).

It *should* work to have a program output a Newick Standard Format tree as ***.nsf** but I have not tested that.

Phylip tree-drawing is also implemented. See **rje_tree** module documentation for details.

3.3.3. Wrapper scripts

If the chosen program does not accept the same input/output commands/formats then a wrapper script should be written. It is suggested to use Perl or Python for this. Although I cannot promise help in every suggestion, you are welcome to e-mail me for help with this and I will see what I can do.

3.3.4. Incorporating Other Programs into the Python Code

If you are feeling brave, you can actually edit the Python modules themselves. The key methods for this are **rje_seq.muscleAln()**, **rje_seq.clustalAln()** and **rje_tree.makeTree()**. Obviously, I cannot promise to give technical support for any changes that are made but, if you know what you are doing, you should be OK and I will help where I can.

3.4. References

- Altschul SF, Gish W, Miller W, Myers EW and Lipman DJ (1990). Basic local alignment search tool. *J Mol Biol*, **215**: 403-410.
- Edgar RC (2004). MUSCLE: a multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics*, **5**: 113.
- Higgins DG and Sharp PM (1988). CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. *Gene*, **73**: 237-244.
- Pearson WR (1994). Using the FASTA program to search protein and DNA sequence databases. *Methods Mol Biol*, **24**: 307-331.
- Pearson WR (2000). Flexible sequence similarity searching with the FASTA3 program package. *Methods Mol Biol*, **132**: 185-219.
- Thompson JD, Higgins DG and Gibson TJ (1994). CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res*, **22**: 4673-4680.