Coding Requirements for Odyssey Functions

on G-language Genome Analysis Environment Version 1

1. AUTHOR

Kazuharu Arakawa, gaou@g-language.org

2. LAST UPDATE OF THIS DOCUMENT

April 1, 2002

3. INTRODUCTION

Odyssey is the code name for the middle layer of G-language GAE, which is a package of Perl subroutines each responsible for a set of analysis. This layer is easily pluggable, and we hope to accumulate useful analysis functions in order to prevent redundancy in development, and to make use of existing tools. The functions can be anything: it could be a method to calculate nucleotide contents, or it can be an interface to pre-existing software. We encourage the users of our software package to submit new analysis subroutines to be packaged with G-language GAE. This documentation explains the coding requirements for the development of Odyssey functions.

4. SYNOPSIS

```
use G;
use SubOpt;
use G::Messenger;

&odyssey($gb);                          #example for regular usage
&odyssey($gb, -output=>’f’);            #example for usage with option
&odyssey(“atgcatgc”);                   #example for opt_as_gb()
        #Options should be parsed with SubOpt module. The only required argument
        #should be the G-language instance or the sequence retrieved with opt_as_gb()
        #All output must use G::Messenger module.
```

```
 sub odyssey{
      opt_default(filename=>’odyssey.png’, output=>’show’);
      my @args = opt_get(@_);
      my $gb = opt_as_gb(shift @args);
         #Options should include “output” (value = “f” for file, “g” for graphics,
         #“show” for automatic display of graphics) and “filename”
      my $output = opt_val(“output”);
      my $filename = opt_val(“filename”);
         #Option values should be kept in local variables, and not called upon usage.

      my @x-axis = (10,20,30,40); my @y1 = (3,8,2,10); my @y2 = (2,13,4,8);
      my $n = 0;
      for ($n = 0; $n <= 3, $n ++){
         msg_send($x_axis[$n] , “,”, $y1[$n], “,”, $y2[$n]);
      }
         #your program should replace the above.

         #Enable different output
      if ($output eq ‘g’ || $output eq ‘show’){
         mkdir (‘graph’, 0777);
         G::Tools::Graph::_UniMultiGrapher(¥@x-axis, ¥@y1, ¥@y2,
                                                    -filename=>’graph/’ . $filename);
         msg_gimv(‘graph/’ . $filename) if ($output eq ‘show’);
      }elsif ($output eq ‘f’){
         mkdir (‘data’, 0777);
         $filename =~ s/¥.png$/¥.csv$/;
         open(OUT, ‘>data/’ . $filename);
         for ($n = 0; $n <= 3, $n ++){
                  printf OUT “%d, %d, %d¥n”, $x_axis[$n], $y1[$n], $y2[$n] ;
         }
         close(OUT);
      }

      return(@y1);
}
```

5. DESCRIPTION

An Odyssey function is a subroutine of analysis. Therefore, if you have a subroutine in Perl, it can easily be converted into an Odyssey function. The following list explains the requirements and recommendations in the conversion process. It is strongly recommended to take a look at the source codes of Odyssey functions for examples.

1. **Enable options for flexibility with SubOpt module. (required)**

   Use SubOpt module for subroutine input, so that the options can be specified. SubOpt module is a set of API that have a similar function as the Perl GetOpt module, except the fact that the options are for subroutines and not for the program itself. See the Manual for SubOpt for details. The basic idea is to set the default options using opt_default(), parsing the @_ arguments with opt_get(), and retrieving the option values using opt_val(). If the function only requires the sequence information and not its annotation, using opt_as_gb() will enable automatic retrieval of sequence information from one of the following: G-language instance, sequence as a scalar, or sequence as a reference.

   The advantages of SubOpt is the following:

   a. Easy way to handle options in subroutines
   b. Uniform way throughout the Odyssey functions
   c. Default values can be specified so that the options can be eliminated
   d. G-language as a system can track what is currently calculated

2. **Default values for options should be set so that the function can be used just by entering the G-language instance. (highly recommended)**

   Ease of use is a key feature of the G-language GAE, and in this sense, any Odyssey function should work by entering one G-language instance (eg. &odyssey($gb)). Every other arguments should be supplied as options, and the default values should be set so that the option can be eliminated. If a function require something that is not a G-language instance as a necessary argument, the name of the function should start with an under-bar '_'.

3. **<u>Option should include "output" and "filename". (highly recommended)</u>**

   All Odyssey function should be able to produce regular STDOUT output, file output, and graphical output if applicable. To manipulate this feature, an option called "output" and "filename" should be present for this purpose, as well as the corresponding function. The recommended default value for the "filename" option is <name of the function>.png for graphics, <name of the function>.csv or .txt for raw data. The value for "output" option should be set to "f" for file output, "g" for graphic output, and "show" for graphic output and automatic display with gimv (G-language IMage Viewer). All graphical output should go to 'graph' directory, and all file output should go to 'data' directory.

4. **<u>All output should use G::Messenger. (required)</u>**

   Use G::Messenger module for output, so that the output can be automatically passed to correct interfaces. See the Manual for G::Messenger for details. The basic idea is to use msg_send() instead of 'print', msg_error() instead of 'print STDERR', and msg_gimv() for display of graphics with gimv.

   The advantages of G::Messenger is the following:

   e. Easy way to handle messages in subroutines
   f. Uniform way throughout the Odyssey functions
   g. Output is passed to correct interfaces (the same program can be used in CUI, GUI, and in Web Interface)
   h. G-language as a system can track what is currently calculated

5. **<u>One analysis should be comprised of one subroutine. (recommended)</u>**

   It is ideal to have compact subroutines whereever possible, but if it is necessary to have multiple subroutines for one analysis, the name of the subroutine required from the main subroutine should start with an under-bar '_' and the name should be as specific as possible.

6. **<u>Use G::Tools::Graph::_UniMultiGrapher for graphing. (recommended)</u>**

   When making graphs, use G::Tools::Graph::_UniMultiGrapher() when applicable. This function produces different types of graphs using gnuplot with very simple options. See the documentation for G::Tools::Graph::_UniMultiGrapher() for details.

7. **<u>Make it as clear, compact, organized, neat, and generic as possible.</u>**

(recommended)

Use strict compile mode to avoid trivial errors, and make the program as neat as possible. Remember that the Odyssey functions are used by every user of G-language GAE.