DOI: 10.1371/journal.pone.0048053

SEED Servers: High-Performance Access to the SEED Genomes, Annotations, and Metabolic Models

Supporting Information S1

Examples of programming using the SEED servers (coded in Perl)

Example Code 1:

URL: http://servers.theseed.org/sapling/server.cgi?code=server_paper_example1.pl

```
1.    use strict;
2.    use SAPserver;
3.    use Data::Dumper;
4.
5.    # Example 1: ACH Data for a Protein
6.    #
7.    # This script takes a protein ID on the command line and
8.    # finds all identifiers in the Sapling database that correspond
9.    # to the identified protein (that is, they have the same amino
10.   # acid sequence). For each identifier found, the following six
11.   # columns will be output.
12.   #
13.   #   1. The identifier found.
14.   #   2. The scientific name of the associated genome (if any).
15.   #   3. 1 if we believe the identifier corresponds to the
16.   #      exact gene identified by the input identifier, else
17.   #      0. If the input identifier does not specify a
18.   #      particular gene, this column will always be 0.
19.   #   4. The functional assignment associated with the protein
20.   #      ID.
21.   #   5. The source of the assignment.
22.   #   6. 1 if the assignment is considered expert, else 0.
23.   #
24.   # The data in columns (1), (4), (5), and (6) are provided
25.   # automatically by the Sapling Server method "equiv_sequence".
26.   # The item in column (3) is extracted via a call to the Sapling
27.   # Server method "genome_names". The tricky part is the value in
28.   # column (3). To make this determination, we make an initial call
29.   # to the "equiv_precise" method to get identifiers for precisely-
30.   # equivalent genes and put them into a hash. The value of
31.   # column (3) is then determined by whether or not an
32.   # identifier is in the precise-equivalence hash.
33.
34.   my $sapObject = SAPserver->new();
35.   my $id = $ARGV[0];
36.   if (! $id) {
37.       die "No protein ID specified.";
```

DOI: 10.1371/journal.pone.0048053

```
38.   } else {
39.       my %preciseHash;
40.       my $precise_assertions_list = $sapObject->equiv_precise_assertions(-ids
=> $id);
41.       $precise_assertions_list = $precise_assertions_list->{$id};
42.       if (@$precise_assertions_list > 0) {
43.         my $inputID = $id;
44.           for my $precise_assertion (@$precise_assertions_list) {
45.               my ($newID, $function, $source, $expert) = @$precise_assertion;
46.               $preciseHash{$newID} = 1;
47.           }
48.       }
49.
50.       my $assertions = $sapObject->equiv_sequence_assertions(-ids => $id);
51.       $assertions = $assertions->{$id};
52.       if (@$assertions < 1) {
53.           print STDERR "No results found.\n";
54.       } else {
55.           for my $assertion (@$assertions) {
56.               my ($newID, $function, $source, $expert, $genomeName) =
@$assertion;
57.               $genomeName = '' if ! defined $genomeName;
58.               my $column3 = ($preciseHash{$newID} ? 1 : 0);
59.               print join("\t", $newID, $genomeName, $column3, $function,
$source,
60.                                   $expert) . "\n";
61.           }
62.       }
63.   }
64.
```

DOI: 10.1371/journal.pone.0048053

Example Code 2:

URL: http://servers.theseed.org/sapling/server.cgi?code=server_paper_example2.pl

```
1.    use strict;
2.    use ANNOserver;
3.
4.    # Example 2: Metabolic Reconstruction for a Complete Prokaryotic Genome
5.
6.    # This script reads a set of gene ID / function pairs and produces a table
of the
7.    # subsystems that can be identified. The pairs are taken from the standard
8.    # input. The input is in the form of a tab-delimited file, each record
9.    # containing a gene ID followed by the function assigned to the gene. The
output
10.   # file is also tab-delimited. Each output record will contain a subsystem
ID,
11.   # its variant code, the relevant functional role, and its gene ID.
12.   #
13.   # By passing in all of the functional roles for a particular genome, you can
14.   # use this program to find roles that may imply missing gene calls.
15.
16.
17.   my $asObject = ANNOserver->new();
18.   my @pairs;
19.   while () {
20.       chomp;
21.       my ($id, $role) = split(/\t/, $_);
22.       push @pairs, [$role, $id];
23.   }
24.
25.   my $reconstruction = $asObject->metabolic_reconstruction(-roles => \@pairs);
26.   for my $record (@$reconstruction) {
27.       my ($variantID, $role, $id) = @$record;
28.       my ($subsysID, $code) = $variantID =~ /^(.+):(.+)$/;
29.       print join("\t", $subsysID, $code, $role, $id) . "\n";
30.   }
```

DOI: 10.1371/journal.pone.0048053

Example Code 3:

URL: http://servers.theseed.org/sapling/server.cgi?code=server_paper_example3.pl

```
1.    use strict;
2.    use SAPserver;
3.    use Data::Dumper;
4.
5.    my ($geneID,$max_distance);
6.    my $usage = "usage: server_paper_example3 GeneID MaxDistance";
7.    (
8.     ($geneID       = shift @ARGV) &&
9.     ($max_distance = shift @ARGV)
10.    )
11.       || die $usage;
12.
13.   my $sapObject = SAPserver->new();
14.   my $geneLocH  = $sapObject->fid_locations(-ids => [$geneID]);
15.   my $geneLoc   = $geneLocH->{$geneID}->[0];
16.   if ($geneLoc =~ /^(\S+)_(\d+)([+-])(\d+)/)  # retrieve an encoded location
17.   {
18.       my($contig,$beg,$strand,$length) = ($1,$2,$3,$4);
19.       my ($left,$right);
20.       if ($strand eq "+")
21.       {
22.         ($left,$right) = ($beg, $beg + ($length-1));
23.       }
24.       else
25.       {
26.         ($left,$right) = ($beg, $beg - ($length-1));
27.       }
28.       my $paddedLeft      = ($left > $max_distance) ? $left  - $max_distance :
1;
29.       my $paddedRight     = $right + $max_distance;
30.       my $sz              = ($paddedRight + 1) - $paddedLeft;
31.       my $region          = $contig . "_" . $paddedLeft . "+" . $sz;
32.       my $genesInRegionH  = $sapObject->genes_in_region(-locations =>
[$region],
33.                                                         -includeLocation =>
1);
34.       my $genesInRegion   = $genesInRegionH->{$region};
35.       foreach my $geneID2 (keys(%$genesInRegion)) {
36.           my $location = $genesInRegion->{$geneID2};
37.           $location =~ /^(\S+)_(\d+)([+-])(\d+)/;
38.         print "$geneID2\t$1\t$2\t$3\t$4\n";
39.       }
40.   }
```

DOI: 10.1371/journal.pone.0048053

Example Code 4:

URL: http://servers.theseed.org/sapling/server.cgi?code=server_paper_example4.pl

```
1.    #!/usr/bin/perl -w
2.    use strict;
3.    use SAPserver;
4.    use COserver;
5.    use SeedUtils;
6.
7.    my $sapObject = SAPserver->new();
8.
9.    my $genomeHash = $sapObject->all_genomes(-complete => 1);
10.   for my $genome (keys %$genomeHash) {
11.       my $genomeName = $genomeHash->{$genome};
12.       my $geneHash = $sapObject->feature_assignments(-genome => $genome,
13.                                                      -type => 'peg');
14.       my @hypotheticalGenes = grep { &SeedUtils::hypo($geneHash->{$_}) } sort
keys %$geneHash;
15.       my $couplingHash = $sapObject->conserved_in_neighborhood(-ids =>
\@hypotheticalGenes,
16.                                                                -hash => 1);
17.       for my $gene (@hypotheticalGenes) {
18.           my $couplingList = $couplingHash->{$gene};
19.           if (defined $couplingList) {
20.               my $subHash = $sapObject->ids_to_subsystems(-ids => [ map { $_-
>[1]} @$couplingList ]);
21.               my ($bestCoupler, $bestScore, $bestFunction) = (undef, 0, '');
22.               for my $coupling (@$couplingList) {
23.                   my ($score, $coupler, $function) = @$coupling;
24.                   if ($subHash->{$coupler} && $score > $bestScore) {
25.                       $bestCoupler = $coupler;
26.                       $bestScore = $score;
27.                       $bestFunction = $function;
28.                   }
29.               }
30.               if (defined $bestCoupler) {
31.                   print join("\t", $gene, $geneHash->{$gene}, $genome,
$genomeName,
32.                                    $bestCoupler, $bestScore, $bestFunction) .
"\n";
33.               }
34.           }
35.       }
36.   }
```

DOI: 10.1371/journal.pone.0048053

Example Code 5:

URL: http://servers.theseed.org/sapling/server.cgi?code=server_paper_example5.pl

```
1.    #!/usr/bin/perl -w
2.    use strict;
3.
4.    use ANNOserver;
5.
6.    # Parms are FASTA input file name, genus, species.
7.    # Produce geneID, type, contig, begin, strand, len on STDOUT.
8.
9.    my $annoObject = ANNOserver->new();
10.   my ($file, $genus, $species) = @ARGV;
11.
12.   open FASTAIN, "<$file";
13.   my $geneCalls = $annoObject->call_genes(\*FASTAIN);
14.   # We throw away the FASTA string returned by call_genes.
15.   my (undef, $geneList) = @$geneCalls;
16.   for my $geneData (@$geneList) {
17.       my ($gene, $contig, $begin, $strand, $len) = @$geneData;
18.       print join("\t", $gene, 'peg', $contig, $begin, $strand, $len);
19.   }
20.   close FASTAIN;
21.   open FASTAIN, "<$file";
22.   my $rnaCalls = $annoObject->find_rnas(\*FASTAIN, $genus, $species,
'Bacteria');
23.   # We throw away the FASTA string returned by find_rnas.
24.   my (undef, $rnaList) = @$rnaCalls;
25.   for my $rnaData (@$rnaList) {
26.       my ($rna, $contig, $begin, $strand, $len) = @$rnaData;
27.       print join("\t", $rna, 'rna', $contig, $begin, $strand, $len);
28.   }
```

DOI: 10.1371/journal.pone.0048053

Example Code 6:

URL: http://servers.theseed.org/sapling/server.cgi?code=server_paper_example6.pl

```
1.    #!/usr/bin/perl -w
2.    use strict;
3.
4.    use ANNOserver;
5.    use Data::Dumper;
6.
7.    # Reads FASTA protein sequences from STDIN.
8.    # Produces sequence ID, role, genome set name, confidence
9.    # If "-blast" is specified, uses BLAST option.
10.
11.   my $annoObject = ANNOserver->new();
12.   my $blastOption = ($ARGV[0] && $ARGV[0] =~ /^-blast/i ? 1 : 0);
13.
14.   my $results = $annoObject->assign_function_to_prot(-input => \*STDIN,
15.                                                       -kmer => 8,
16.                                                       -scoreThreshold => 3,
17.                                                       -assignToAll =>
$blastOption);
18.   while (my $data = $results->get_next()) {
19.       my ($id, $role, $genomeSet, undef, $hits) = @$data;
20.       # Only proceed if a role was found.
21.       if ($role) {
22.           $genomeSet = '' if ! defined $genomeSet;
23.           print join("\t", $id, $role, $genomeSet, $hits) . "\n";
24.       }
25.   }
```

DOI: 10.1371/journal.pone.0048053

Example Code 7:

URL: http://servers.theseed.org/sapling/server.cgi?code=server_paper_example7.pl

```
1.    #!/usr/bin/perl -w
2.    use strict;
3.    use FBAMODELserver;
4.
5.    # Reads the SEED genome-scale metabolic model of E. coli
6.    # Runs flux variability analysis, flux balance analysis, and single gene
knockout simulations in E. coli
7.    # Prints all results to file
8.
9.    #1.) Creating perl object which interfaces with the SEED server API
10.   my $fbaObject = FBAMODELserver->new();
11.
12.   #2.) Obtain a list of the reaction IDs in the E. coli model
13.   my $reactionList = $fbaObject->get_reaction_id_list({id => "Seed83333.1"});
14.
15.   #3.) Obtain a list of the compound IDs in the E. coli model
16.   my $compoundList = $fbaObject->get_compound_id_list({id => "Seed83333.1"});
17.
18.   #4.) Obtaining detailed data on E. coli reactions
19.   my $reactionData = $fbaObject->get_reaction_data({"id" => $reactionList-
>{"Seed83333.1"},"model" => ["Seed83333.1"]});
20.
21.   #5.) Obtaining detailed data on E. coli compounds
22.   my $compoundData = $fbaObject->get_compound_data({"id" => $compoundList-
>{"Seed83333.1"},"model" => ["Seed83333.1"]});
23.
24.   #6.) Running flux variability analysis on E. coli model while simulating
growth in LB media
25.   my $fvaOutput = $fbaObject->classify_model_entities({"parameters" =>
[{id=>"Seed83333.1",media=>"ArgonneLBMedia"}]});
26.   #Placing fva output into a hash to simplify access
27.   my $fvaHash;
28.   for (my $i=0; $i < @{$fvaOutput->[0]->{entities}};$i++) {
29.         $fvaHash->{substr($fvaOutput->[0]->{entities}->[$i],0,8)} =
$fvaOutput->[0]->{classes}->[$i];
30.   }
31.
32.   #7.) Simulating growth of E. coli model in LB media
33.   my $fbaOutput = $fbaObject->simulate_model_growth({"parameters" =>
[{id=>"Seed83333.1",media=>"Carbon-D-Glucose"}]});
34.   #Placing fba output into a hash to simplify access
35.   my $fbaHash;
36.   for (my $i=0; $i < @{$fbaOutput->[0]->{entities}};$i++) {
37.         $fbaHash->{$fbaOutput->[0]->{entities}->[$i]} = $fbaOutput->[0]-
>{fluxes}->[$i];
38.   }
39.
40.   #8.) Simulating single gene knockout in E. coli model during growth in LB
media
41.   my $KOOutput = $fbaObject->simulate_all_single_gene_knockout({"parameters"
=> [{id=>"Seed83333.1",media=>"ArgonneLBMedia"}]});
42.
43.   #9.) Printing all compound data to output file CompoundTbl.txt
```

DOI: 10.1371/journal.pone.0048053

```
44.   my $columns = ["DATABASE","NAME","FORMULA","CHARGE","FVA class","FBA uptake
flux"];
45.   open (OUTPUT, ">CompoundTbl.txt");
46.   print OUTPUT join("\t",@{$columns})."\n";
47.   for (my $i=0; $i < @{$compoundList->{"Seed83333.1"}}; $i++) {
48.         for (my $j=0; $j < @{$columns}; $j++) {
49.               if ($j > 0) {
50.                     print OUTPUT "\t";
51.               }
52.               if ($columns->[$j] eq "FVA class") {
53.                     if (defined($fvaHash->{$compoundList->{"Seed83333.1"}-
>[$i]})) {
54.                           print OUTPUT $fvaHash->{$compoundList-
>{"Seed83333.1"}->[$i]};
55.                     }
56.               } elsif ($columns->[$j] eq "FBA uptake flux") {
57.                     if (defined($fbaHash->{$compoundList->{"Seed83333.1"}-
>[$i]})) {
58.                           print OUTPUT $fbaHash->{$compoundList-
>{"Seed83333.1"}->[$i]};
59.                     }
60.               } else {
61.                     if (defined($compoundData->{$compoundList-
>{"Seed83333.1"}->[$i]}->{$columns->[$j]})) {
62.                           print OUTPUT join(", ",@{$compoundData-
>{$compoundList->{"Seed83333.1"}->[$i]}->{$columns->[$j]}});
63.                     }
64.               }
65.         }
66.         print OUTPUT "\n";
67.   }
68.   close (OUTPUT);
69.
70.   #10.) Printing all reaction data to output file ReactionTbl.txt
71.   $columns = ["DATABASE","NAME","EQUATION","Seed83333.1
COMPARTMENT","Seed83333.1 ASSOCIATED PEG","Seed83333.1 DIRECTIONALITY","FVA
class","FBA flux"];
72.   open (OUTPUT, ">ReactionTbl.txt");
73.   print OUTPUT join("\t",@{$columns})."\n";
74.   for (my $i=0; $i < @{$reactionList->{"Seed83333.1"}}; $i++) {
75.         for (my $j=0; $j < @{$columns}; $j++) {
76.               if ($j > 0) {
77.                     print OUTPUT "\t";
78.               }
79.               if ($columns->[$j] =~ m/^Seed83333\.1\s(.+)/) {
80.                     my $columnName = $1;
81.                     if (defined($reactionData->{$reactionList-
>{"Seed83333.1"}->[$i]}->{"Seed83333.1"}->{$columnName})) {
82.                           print OUTPUT join(" or ",@{$reactionData-
>{$reactionList->{"Seed83333.1"}->[$i]}->{"Seed83333.1"}->{$columnName}});
83.                     }
84.               } elsif ($columns->[$j] eq "FVA class") {
85.                     if (defined($fvaHash->{$reactionList->{"Seed83333.1"}-
>[$i]})) {
86.                           print OUTPUT $fvaHash->{$reactionList-
>{"Seed83333.1"}->[$i]};
87.                     }
88.               } elsif ($columns->[$j] eq "FBA flux") {
```

DOI: 10.1371/journal.pone.0048053

```
89.                     if (defined($fbaHash->{$reactionList->{"Seed83333.1"}-
>[$i]})) {
90.                           print OUTPUT $fbaHash->{$reactionList-
>{"Seed83333.1"}->[$i]};
91.                     }
92.                } else {
93.                     print OUTPUT $reactionData->{$reactionList-
>{"Seed83333.1"}->[$i]}->{$columns->[$j]}->[0];
94.                }
95.          }
96.          print OUTPUT "\n";
97.    }
98.    close (OUTPUT);
99.
100.   #11.) Printing essential gene predictions to output file GeneTbl.txt
101.   open (OUTPUT, ">GeneTbl.txt");
102.   print OUTPUT "Predicted essential E. coli genes\n";
103.   for (my $i=0; $i < @{$KOOutput->[0]->{"essential genes"}}; $i++) {
104.         print OUTPUT $KOOutput->[0]->{"essential genes"}->[$i]."\n";
105.   }
106.   close (OUTPUT);
```

DOI: 10.1371/journal.pone.0048053

Example Code 8:

Short URL: http://bit.ly/server_paper_example8

This example includes two scripts:

(i) parse_uniprot.pl

URL: http://edwards-sdsu.cvs.sourceforge.net/viewvc/edwards-sdsu/bioinformatics/bin/parse_uniprot.pl?view=log

```
1     #__perl__
2
3     use strict;
4     use Digest::MD5;
5     use Data::Dumper;
6
7     # a simple uniprot parser because the parser in the version of bioperl  I
have is broken.
8     # this is extremely simple, we are just printing out the id, accession
number, go terms, md5 sum (so we can map to seed ids) and protein sequence
9
10    my $f = shift || die "Swiss Prot file?";
11    my $record;
12    my $inseq = 0;
13    # open the file, regardless of format
14    if ($f =~ /gz$/) {
15            open(IN, "gunzip -c $f|") || die "Can't open a pipe to gunzip $f";
16    } elsif ($f =~ /zip$/) {
17            open(IN, "unzip -p $f|") || die "can't open a pipe to unzip $f";
18    }       else {
19            open(IN, $f) || die "can't open $f";
20    }
21
22    while (<IN>) {
23            chomp;
24            if (/^\/\/$/) {
25                    if ($record) {&output($record)};
26                    undef $record;
27                    $inseq=0;
28                    next;
29            }
30            if ($inseq) {
31                    s/^\s+//;
32                    $record->{'SQ'} .= $_;
33                    next;
34            }
35            m/^(\w\w)\s+(.*?)$/;
36            my ($id, $data) = ($1, $2);
37            if ($id eq "SQ") {$inseq = 1; next}
38            push @{$record->{$id}}, $data;
39    }
40
41
42    sub output {
43            my $rec = shift;
44            #die Dumper($rec);
```

DOI: 10.1371/journal.pone.0048053

```
45
46              my $trans = $rec->{'SQ'};
47              $trans =~ s/\s//g;
48              my $md5 = Digest::MD5::md5_hex(uc($trans));
49              my $go = join("; ", grep {m/^GO/} @{$rec->{'DR'}});
50              $rec->{'ID'}->[0] =~ m/^\s*(\S+)\s/;
51              my $id = $1;
52
53              foreach my $ac (map {split /\;\s*/, $_} @{$rec->{'AC'}}) {
54                      print join("\t", $id, $ac, $go, $md5, $trans), "\n";
55              }
56      }
57
```

(ii) map_uniprot_to_seed.pl

URL: http://edwards-sdsu.cvs.sourceforge.net/viewvc/edwards-sdsu/bioinformatics/bin/map_uniprot_to_seed.pl?view=log

```
1       #__perl
2
3       use strict;
4       use SAPserver;
5       use Data::Dumper;
6       my $sap=new SAPserver;
7       use Getopt::Std;
8       my %opts;
9       getopts('f:n:', \%opts);
10
11      die <<EOF unless ($opts{f});
12      $0
13      -f filename to get
14      -n number of proteins to parse at once. The default is 1,000 proteins
15
16
17      This is a program to retrieve essential seed data (role, subsystems, and
subsystem classifications) about a set of proteins. To map into the seed we map
based on the MD5 sum of the uppercase protein sequence (just the protein sequence
alone, and without any identifiers or stop symbol). We can then move from those
ids to subsystems and roles.
18
19      This code uses the seed servers, and you can find more information about
them at http://servers.theseed.org/ You will need to install the seed servers
before starting, but that should install all the commands you need.
20
21      Basically, we make remote calls that retrieve data, and the calls are
designed so that we can retrieve a bolus of data and compute on it. In this case,
you can set how many pieces of data to send/receive with the -n option. I suggest
a relatively large number, like 1000, since the delay is typically in retrieving
the data and not in sending the data over the internet.
22
23      This will print the results out, and so you may need to redirect the output.
24
25      EOF
26
27
28      unless (defined $opts{n}) {$opts{n} = 1000}
29
```

DOI: 10.1371/journal.pone.0048053

```
30    # get the subsytem hierarchy. We only need this once, so getting it here
will let us use it later
31    my $ssHash = $sap->all_subsystems(-usable => 1);
32
33    # counters and raw data
34    my $ids; my $count=0;
35    # open the file and iterate through every line
36    # this allows us to open gzipped or zipped files
37    if ($opts{f} =~ /gz$/) {
38            open(IN, "gunzip -c $opts{f}|") || die "Can't open a pipe to gunzip
$opts{f}";
39    } elsif ($opts{f} =~ /zip$/) {
40            open(IN, "unzip -p $opts{f}|") || die "can't open a pipe to unzip
$opts{f}";
41    }       else {
42            open(IN, $opts{f}) || die "can't open $opts{f}";
43    }
44
45    while (<IN>) {
46            chomp;
47            # this is the input data
48            my ($id, $ac, $go, $md5, $trans)= split /\t/;
49            # store it and make a call with lots of data
50            $ids->{$md5}=[$id, $ac, $go, $md5, $trans];
51            $count++;
52            if ($count == $opts{n}) {
53                    # we need to make the call and print the information out
54                    # don't forget to reset our counters.
55                    &get_data($ids);
56                    $count=0;
57                    $ids={};
58            }
59    }
60    close IN;
61
62
63    sub get_data {
64            my $ids = shift;
65
66            # convert the md5 sums to fig ids
67            # all of the subsequent calls require fig ids
68            my $idHash = $sap->proteins_to_fids({-prots=>[keys %$ids]});
69            my $figids;
70            foreach my $id (keys %$idHash) {
71                    map {$figids->{$_}=1} @{$idHash->{$id}};
72            }
73
74            # get the subsystems for each proteins
75            my $subsysHash = $sap->ids_to_subsystems({-ids=>[keys %$figids], -
usable=>1});
76
77            # now print out everything
78            # we need to iterate through the md5s and the fid ids to merge
everything together
79            foreach my $md5 (keys %$ids) {
80                    foreach my $fid (@{$idHash->{$md5}}) {
81                            # get the functional role and subsystem for this
protein
```

DOI: 10.1371/journal.pone.0048053

```
82                              # recall that this is a one:many relationship
83                              foreach my $tple (@{$subsysHash->{$fid}}) {
84                                      my ($role ,$ss)=@$tple;
85                                      # in case the classification isn't defined,
we'll make it null
86                                      if (!defined $ssHash->{$ss}) {$ssHash-
>{$ss}=['', ['', '']]}
87                                      # print everything out
88                                      print join("\t", @{$ids->{$md5}}, $fid,
$role, $ss, @{$ssHash->{$ss}->[1]}), "\n";
89                              }
90                      }
91              }
92      }
93
94
95
```