Chapter V

Creating a Gene Prediction and BLAST Analysis Pipeline

Introduction

Gene prediction and *gene annotation* are fundamental aspects of genome-sequencing projects and discovery research. These activities involve determination of complete *gene structures* from the raw DNA sequence and attributing functions to them, by way of computational methods, at least as a first step. These methods try to implement an understanding of the way in which the structural elements such as *coding*, *non-coding* and *regulatory elements* are organized within genes, to extract meaningful information from raw nucleotide sequences.

Gene prediction programs, specifically, are designed to recognize genetic signals that are embedded in DNA sequences to make predictions about gene structure. We will explore gene prediction programs in more detail in this Chapter and build an analytic pipeline that will tie gene prediction and the BLAST application we built in earlier Chapters.

Gene Prediction Programs

Gene prediction methods that rely only on information that is encoded in the sequence itself to make predictions are called *ab initio* (Latin: from the beginning) methods. These methods use signals within DNA such as *splice sites*, *start* and *stop codons*, *promoters* and *terminators* of *transcription*, *polyadenylation sites*, *ribosomal binding sites*, *CpG islands*, and various *transcription factor binding sites* to predict the presence of *exons*. ab initio methods such as *Genscan* rely on probabilistic models known as *Hidden Markov models* (HMMs) to discern patterns within DNA. An HMM models the different states that a DNA sequence can exist in and the transition probabilities between the states. The different states of DNA are the ones enumerated above such as promoter, *intron*, *exon* etc. The term 'Hidden' comes from the fact that the sequence itself is visible but the states are hidden.

DNA Transcription and Translation

Although a detailed treatment of these subjects are out of the scope of this book, an introduction of the basic concepts in essential to understand the biology and behavior of the DNA and RNA. We had mentioned the terms transcription and translation in the last section. Transcription is the process by which a DNA molecule is copied into an RNA molecule, while translation is the process by which the RNA sequence is used by the cellular machinery to synthesize proteins.

Transcription may result in one of three types of RNA: Messenger RNA (mRNA), transfer RNAs (tRNA) or ribosomal RNA (rRNA). mRNA molecules serve as 'messengers' that specify the code for the synthesis of amino acids (during translation) and therefore the name messenger RNA. tRNAs form covalent attachments to individual amino acids and recognize the encoded sequences of the mRNAs to allow correct insertion of amino acids into the elongating polypeptide chain during translation. rRNAs are assembled together with numerous proteins to form complexes known as ribosomes. Ribosomes engage mRNAs and form a catalytic domain into which the tRNAs enter with their attached amino acids. The proteins of the ribosomes catalyze all of the functions of polypeptide synthesis.

During the process of transcription, the DNA double helix unwinds and one strand serves as the template for the synthesis of the RNA strand. Either strand can serve as the template - which strand becomes the template depends on a combination of transcription initiation and termination signals such as promoter and enhancer sequences that are present on the DNA. Transcription is actually a polymerization reaction in which individual nucleotides are linked together by an enzymatic reaction (catalyzed by the enzyme *RNA polymerase*) into a chain to form RNA.

In nature, these processes are orchestrated in a finely tuned and regulated manner involving an intricate interplay of a large number of proteins, which recognize specific signals and patterns on the sequences they bind. An example is what are known as *CpG islands*, which are regions within DNA that often occur near the beginning of genes, where the frequency of the *dinucleotide* CG (that is, the nucleotide bases *cytosine* and *guanine*) is more than in the rest of the genome

We had also mentioned exons and introns and these are simply terms used to refer to regions of DNA that code for or don't code for proteins respectively. To elaborate, higher organisms (eukaryotes) have what are called "*split genes*", that is, a large proportion of their genes are not continuous linear entities, but instead may be interrupted throughout their length by sequences that do not code for protein. A piece of DNA may therefore contain coding sequences with intervening non-coding sequences. The intervening non-coding segments are called the introns and do not code for protein. The coding sequences are exons and do code for protein. For example, the Cystic Fibrosis transmembrane regulator (CFTR) gene's coding regions (exons) are scattered over 250,000 base pairs of genomic DNA and is made up of 27 exons. During transcription, introns are removed from the CFTR gene and exons are pieced together by a process known as *RNA splicing* to form a 6100-bp mRNA transcript that is translated into the 1480 amino acid sequence (the CFTR protein). In contrast, the 384 nucleotide human pancreatic ribonuclease gene is intronless and codes for a 128 amino acid protein. A highly schematic view of the RNA splicing process is show in **Fig. 5.1**.

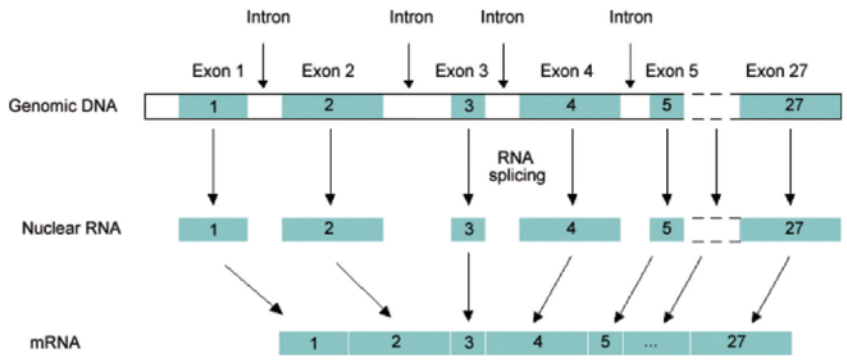


Fig. 5.1: Schematic of RNA splicing

Gene Prediction with Genscan

Genscan is one of the most effective among the many exon prediction programs to date. In this Chapter, we will build an application that will allow users to perform Genscan-based predictions on an unknown piece of DNA and analyze the predicted genes and peptides with BLAST using the SwingBlast application that we wrote earlier. The rationale to combine the two programs into this pipeline is simple – once we know that a newly sequenced stretch of DNA probably contains potential coding regions, we would like to know what peptides they may code for and what functions they perform. As we learned in Chapter 2, a BLASTX analysis of a nucleotide sequence, for example, compares a nucleotide query sequence translated in all reading frames against a protein sequence database and produces matches to known proteins. This information in turn provides clues to the probable function of an unknown peptide sequence. The integrated Genscan and BLAST pipeline can be used to perform such functional characterization of newly sequenced DNA fragments.

Genscan was written by Chris Burge and Samuel Karlin at the Department of Mathematics, Stanford University. Genscan utilizes the same basic signals described earlier to build complete *gene structures* (that is, introns + exons) from human genomic sequences. Specifically, these include transcriptional, translational and splicing signals (including elements present in most eukaryotic promoters such as the *TATA box* and cap site), as well as length distributions and compositional features of exons, introns and *intergenic* regions. Importantly, Genscan also makes use of the many substantial differences in gene density and structure based on GC composition of the human genome. For example, it is known that gene density in GC rich regions is five times higher than in regions with moderate GC content and ten times higher in rich AT rich regions. Four categories of DNA were identified based on their GC content:

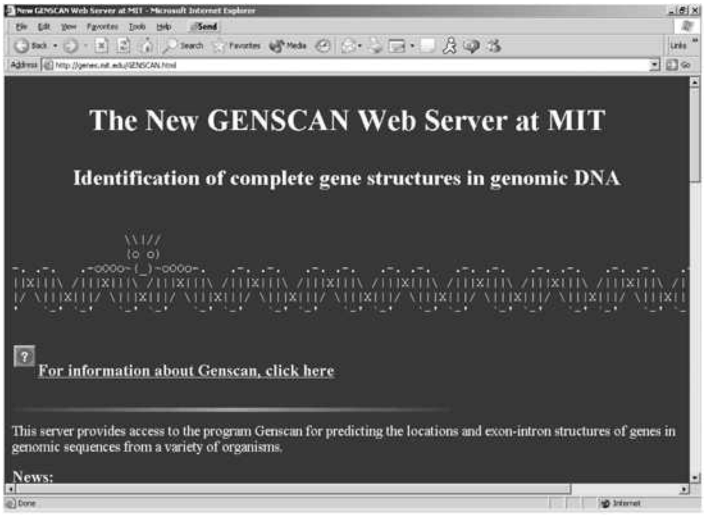
1. < 43% GC
2. 43 -51% GC
3. 51 - 57% GC
4. > 57% GC

These are known as *isochores*. Thus, if the input genomic sequence has a GC content of 45 % it is said to have an isochore value of 2. ab initio programs traditionally have been poor at predicting genes in regions containing multiple genes, especially when present on both DNA strands. Genscan addresses these problems by using an explicitly double-stranded genomic sequence model, which has the likelihood of genes occurring on both DNA strands. Second, while most programs assume the presence of exactly one complete gene in the input sequence, Genscan treats the more general case in which the sequence may contain a partial gene, a complete gene, multiple complete (or partial) genes on either strand, or no gene at all. A significant difference in Genscan also is the incorporation of splice donor signal information based on the mechanism of donor splice site recognition in pre-mRNA sequences by *U1 small nuclear ribonucleoprotein particle* (U1 snRNP).

Running Genscan Analyses

Running and interpreting a Genscan analysis is rather straightforward. Point your browser to the Genscan server at MIT: http://genes.mit.edu/GENSCAN.html (**Fig. 5.2**). For this exercise we will use a 175 kilobase human bacterial artificial chromosome (BAC) with the accession number AC092818 from NCBI. Genscan has been 'trained' to work with vertebrate, arabidopsis and maize sequences (**Fig. 5.3**). Since we are analyzing a human BAC, we choose the vertebrate option. We will use the default sub-optimal exon cut-off value of 1 for our purposes. This value defines the threshold, which determines if exons that do not meet the criteria (sub-optimal exons) will be shown or not.

You can give a sequence name if you are analyzing a large number of sequences and want to label each output by a unique identifier. In this case, we will just use the BAC accession number (**Fig. 5.4**). The program gives an option to print out the predicted proteins or the predicted proteins along with their nucleotide sequences. We will choose the latter option (**Fig. 5.5**).


New GENSCAN Web Server at MIT - Microsoft Internet Explorer
File Edit View Favorites Tools Help Send
Back Search Favorites Media
Address http://genes.mit.edu/GENSCAN.html
Done
Internet

The New GENSCAN Web Server at MIT

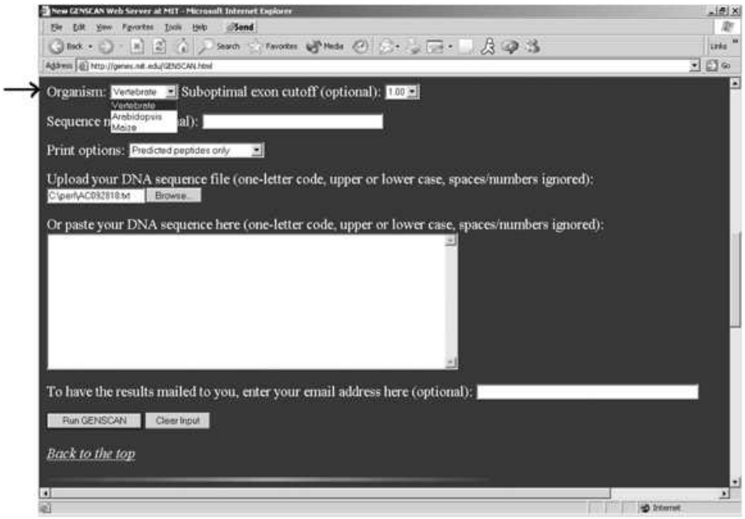
Identification of complete gene structures in genomic DNA

For information about Genscan, click here

This server provides access to the program Genscan for predicting the locations and exon-intron structures of genes in genomic sequences from a variety of organisms.

News:

Fig. 5.2. The Genscan web server


New GENSCAN Web Server at MIT - Microsoft Internet Explorer
File Edit View Favorites Tools Help Send
Back Search Favorites Media
Address http://genes.mit.edu/GENSCAN.html
Vertebrate
Arabidopsis
Maize
Browse...
Run GENSCAN
Clear Input
Internet

Organism: Vertebrate Suboptimal exon cutoff (optional): 1.00

Sequence n[...]al):

Print options: Predicted peptides only

Upload your DNA sequence file (one-letter code, upper or lower case, spaces/numbers ignored):
C:\perl\AC092818.txt

Or paste your DNA sequence here (one-letter code, upper or lower case, spaces/numbers ignored):

To have the results mailed to you, enter your email address here (optional):

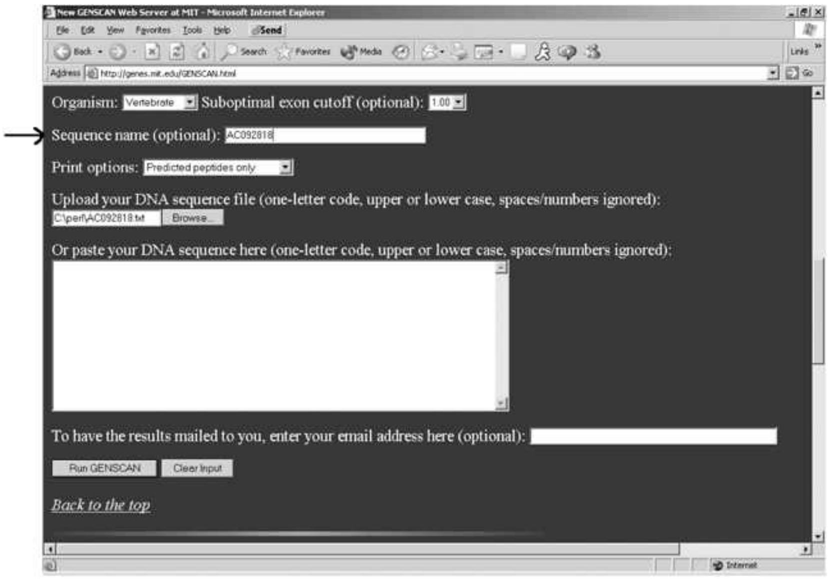
Back to the top

Fig. 5.3. Setting Genscan parameters

The sequence can be either uploaded or pasted directly in the text box. Uploading a sequence is more convenient if you are handling very large sequences, as is the case here (**Fig. 5.5**). Finally, you can specify an email address if you want to receive the results via email. We will hit the "Run Genscan" button and just wait to see the results in the browser. **Fig. 5.6** and **Fig. 5.7** show the results of the Genscan analysis.

Analyzing GenScan Output

The GenScan header gives information on the input sequence and the parameters used such as name, size and isochore classification (categorization based on GC content) of the sequence, and the matrix used for the analysis (HumanIso.smat). The body of the analysis consists of the predicted peptide and the corresponding CDS sequences. As is evident from the output there were eight predicted peptides in this sequence. The complete gene structure of each peptide is listed after the header (**Table 5.1**).


New GENSCAN Web Server at MIT - Microsoft Internet Explorer
File Edit View Favorites Tools Help Send
Back Search Favorites Media
Links
Address http://genes.mit.edu/GENSCAN.html
Browse...
Run GENSCAN
Clear Input
Back to the top
Internet

Organism: Vertebrate Suboptimal exon cutoff (optional): 1.00

Sequence name (optional): AC092818

Print options: Predicted peptides only

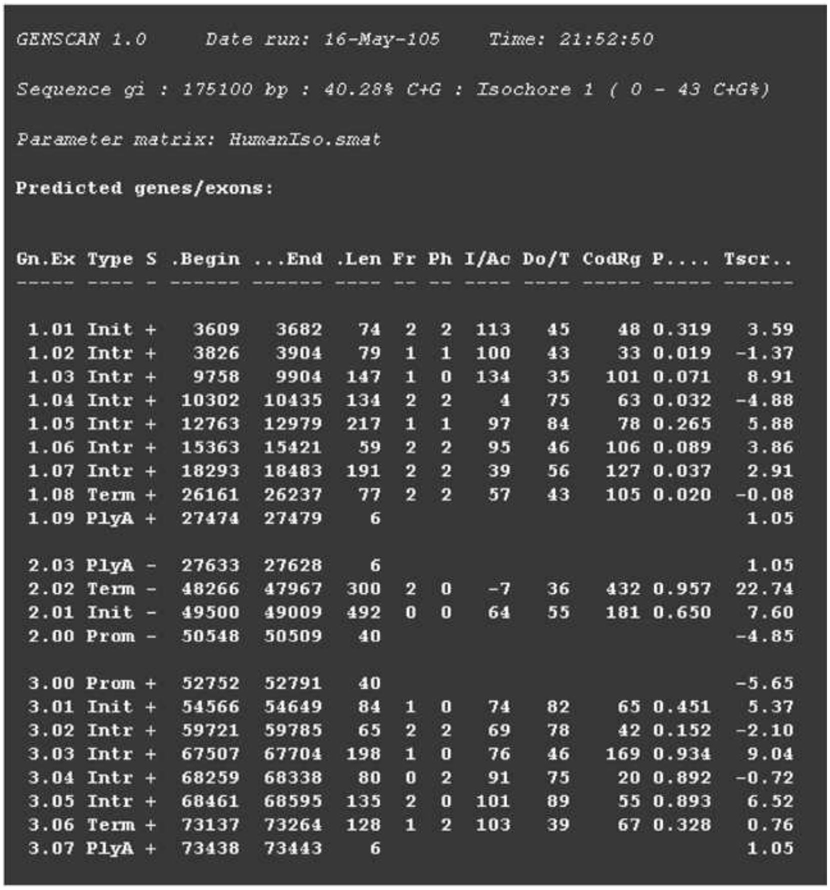
Upload your DNA sequence file (one-letter code, upper or lower case, spaces/numbers ignored):
C:\perl\AC092818.txt

Or paste your DNA sequence here (one-letter code, upper or lower case, spaces/numbers ignored):

To have the results mailed to you, enter your email address here (optional):

Fig. 5.4. Entering an identifier

Table 5.1. Gene structures



```
GENSCAN 1.0     Date run: 16-May-105     Time: 21:52:50

Sequence gi : 175100 bp : 40.28% C+G : Isochore 1 ( 0 - 43 C+G%)

Parameter matrix: HumanIso.smat

Predicted genes/exons:


Gn.Ex Type S .Begin ...End .Len Fr Ph I/Ac Do/T CodRg P.... Tscr..
----- ---- - ------ ------ ---- -- -- ---- ---- ----- ----- ------

 1.01 Init +   3609   3682   74  2  2  113   45    48 0.319   3.59
 1.02 Intr +   3826   3904   79  1  1  100   43    33 0.019  -1.37
 1.03 Intr +   9758   9904  147  1  0  134   35   101 0.071   8.91
 1.04 Intr +  10302  10435  134  2  2    4   75    63 0.032  -4.88
 1.05 Intr +  12763  12979  217  1  1   97   84    78 0.265   5.88
 1.06 Intr +  15363  15421   59  2  2   95   46   106 0.089   3.86
 1.07 Intr +  18293  18483  191  2  2   39   56   127 0.037   2.91
 1.08 Term +  26161  26237   77  2  2   57   43   105 0.020  -0.08
 1.09 PlyA +  27474  27479    6                                1.05

 2.03 PlyA -  27633  27628    6                                1.05
 2.02 Term -  48266  47967  300  2  0   -7   36   432 0.957  22.74
 2.01 Init -  49500  49009  492  0  0   64   55   181 0.650   7.60
 2.00 Prom -  50548  50509   40                               -4.85

 3.00 Prom +  52752  52791   40                               -5.65
 3.01 Init +  54566  54649   84  1  0   74   82    65 0.451   5.37
 3.02 Intr +  59721  59785   65  2  2   69   78    42 0.152  -2.10
 3.03 Intr +  67507  67704  198  1  0   76   46   169 0.934   9.04
 3.04 Intr +  68259  68338   80  0  2   91   75    20 0.892  -0.72
 3.05 Intr +  68461  68595  135  2  0  101   89    55 0.893   6.52
 3.06 Term +  73137  73264  128  1  2  103   39    67 0.328   0.76
 3.07 PlyA +  73438  73443    6                                1.05
```

The most important aspects f this table are the gene and exon number, the type of exon, the strand information (+/-), the background and end positions, the length of each exon in basepairs, the frame and the scores. The key to the abbreviations is provided at the end of the output (**Table 5.2**).

Table 5.2. Abbreviations and explanations

Gn.Ex	gene number, exon number (for reference)

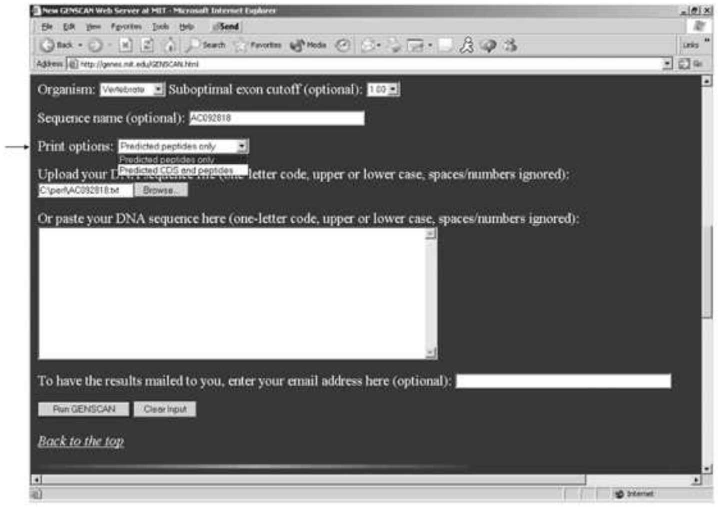
Type	Init = Initial exon (ATG to 5' splice site)
	Intr = Internal exon (3' splice site to 5' splice site)
	Term = Terminal exon (3' splice site to stop codon)
	Sngl = Single-exon gene (ATG to stop)
	Prom = Promoter (TATA box / initation site)
	PlyA = poly-A signal (consensus: AATAAA)

S	DNA strand (+ = input strand; - = opposite strand)
Begin	beginning of exon or signal (numbered on input strand)
End	end point of exon or signal (numbered on input strand)
Len	length of exon or signal (bp)
Fr	reading frame (a forward strand codon ending at x has frame x mod 3)
Ph	net phase of exon (exon length modulo 3)
I/Ac	initiation signal or 3' splice site score (tenth bit units)
Do/T	5' splice site or termination signal score (tenth bit units)
CodRg	coding region score (tenth bit units)
P	probability of exon (sum over all parses containing exon)
Tscr	exon score (depends on length, I/Ac, Do/T and CodRg scores)

Each pair of peptide and CDSs (as shown below for the first set) are in Fasta format and have unique identifiers where the sequences are numbered sequentially.

```
   >gi|GENSCAN_predicted_peptide_1|325_aa
   MALISFTSPFNFIGKKSWQCITEAGFDKVDETIIFVISQSSRNVIVGEFLQDPCQGLPL
L
   KDLSSKQAANLFPWQRMEAVACDILLIMQPGHGQPAFLQGMSSRLSGAAEQVGSWSMRS
Q
   RHSLLWSVPEPVQQAGFLFPEALQSAGCFLPSNIGLQVLQFWTLGLTSVVCQGLSGLWP
Q
   IEGCTVGFSTFEVLGLGLASLLLSLQTAYCGTSPCDHSSSLSDSKAAVLENIGLLPLTH
L
   SECSRGGTQTGISGLKTELGAKVARVCQAEYGGESHAEREFWTPTEESLRVYKRGLISS
A
   SGISVDHGSLPEGLTKTFIPEGYEP

>gi|GENSCAN_predicted_CDS_1|978_bp
atggccctaatcagttttacatctccgtttaattttattggaaagaagagctggcaatgc
atcacagaggccggctttgacaaagtggatgaaacaattatcttcgttatcagccaaagc
agtagaaatgtgatagttggggaattttgcaggacccatgccagggcttacctctgcta
aaggatttgtcctcaaagcaggcagcaaatctgttcccttggcagaggatggaagccgtg
gcttgtgacattctcctgataatgcagccaggccacgggcagccagcatttctgcagggg
atgagctccaggctcagtggggcagcagagcaagtggggagctggtccatgaggagtcag
cgtcattccttgctgtggtctgttcctgaaccagtccaacaggctggcttcctgttccca
gaagccctccaaagtgctggatgcttcctgccatcgaacattggactccaagttcttcag
ttttggactcttggacttacatcagtggtttgccagggactctcaggcctttggcctcag
```

```
attgaaggctgcactgtcggcttctctacttttgaggttttgggactcggactggcttcc
ttgctcctcagcttgcagacagcctattgtgggacttcaccttgtgatcattccagcagc
ctttcggattccaaagcggctgtcctggaaaatatagggctccttccactaacccacctc
tctgaatgcagcagaggtggaacccagacagggatcagtgggttaaagacagagctggga
gccaaggtagccagagtttgccaggcagagtatggcggagagagccacgcagagagagaa
ttctggacacctacggaggaatctcttcgagtatataaaagaggactgatcagcagtgca
tcaggtatctctgttgatcatggttctttacccgaaggactgactaaaacctttattcct
gaagggtatgaaccatag
```


New GENSCAN Web Server at MIT - Microsoft Internet Explorer

Address: http://genes.mit.edu/GENSCAN.html

Organism: Vertebrate Suboptimal exon cutoff (optional): 1.00

Sequence name (optional): AC092818

Print options: Predicted peptides only
Predicted peptides only
Predicted CDS and peptides

Upload your DNA sequence file (one-letter code, upper or lower case, spaces/numbers ignored):
C:\perl\AC092818.txt Browse...

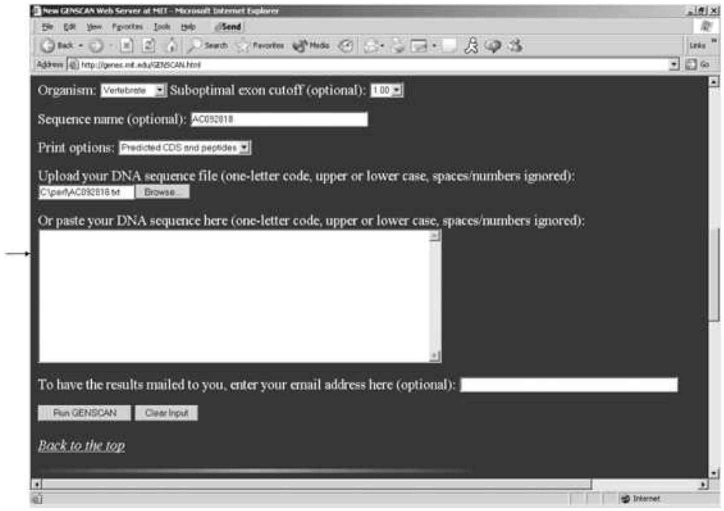
Or paste your DNA sequence here (one-letter code, upper or lower case, spaces/numbers ignored):

To have the results mailed to you, enter your email address here (optional):

Run GENSCAN Clear Input

Back to the top

Fig. 5.5. Printing peptides and the corresponding coding sequences (CDS)


New GENSCAN Web Server at MIT - Microsoft Internet Explorer

Address: http://genes.mit.edu/GENSCAN.html

Organism: Vertebrate Suboptimal exon cutoff (optional): 1.00

Sequence name (optional): AC092818

Print options: Predicted CDS and peptides

Upload your DNA sequence file (one-letter code, upper or lower case, spaces/numbers ignored):
C:\perl\AC092818.txt Browse...

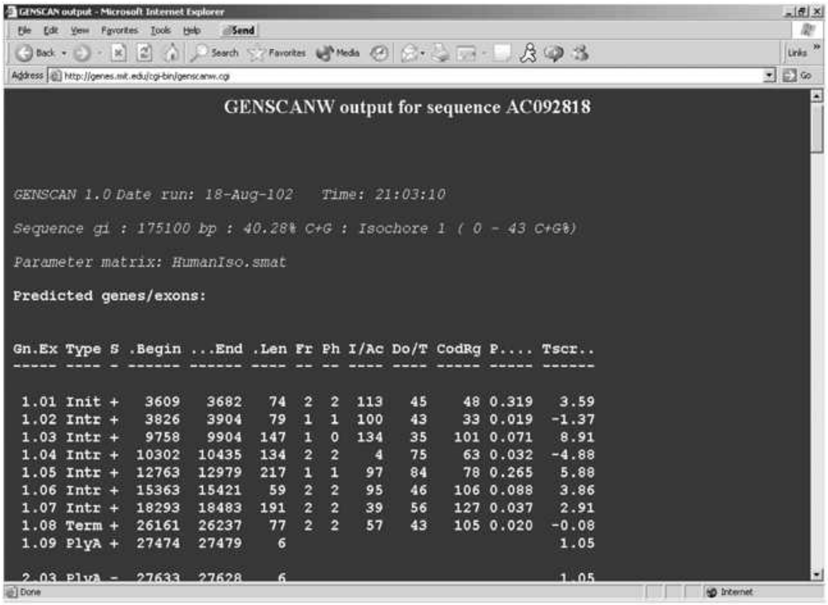
Or paste your DNA sequence here (one-letter code, upper or lower case, spaces/numbers ignored):

To have the results mailed to you, enter your email address here (optional):

Run GENSCAN Clear Input

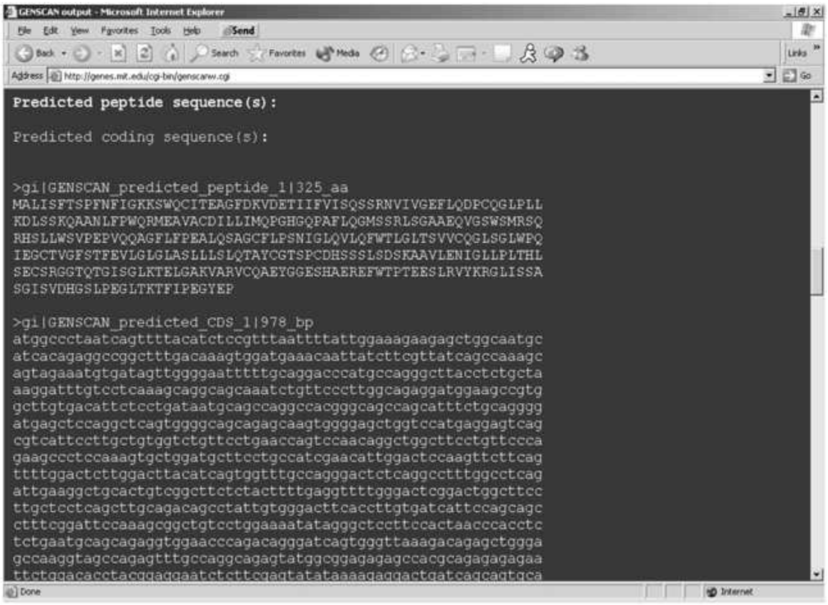
Back to the top

Fig. 5.6. Uploading the BAC sequence



```
GENSCANW output for sequence AC092818

GENSCAN 1.0 Date run: 18-Aug-102   Time: 21:03:10

Sequence gi : 175100 bp : 40.28% C+G : Isochore 1 ( 0 - 43 C+G%)

Parameter matrix: HumanIso.smat

Predicted genes/exons:

Gn.Ex Type S .Begin ...End .Len Fr Ph I/Ac Do/T CodRg P.... Tscr..
----- ---- - ------ ------ ---- -- -- ---- ---- ----- ----- ------

 1.01 Init +   3609   3682   74  2  2  113   45    48 0.319   3.59
 1.02 Intr +   3826   3904   79  1  1  100   43    33 0.019  -1.37
 1.03 Intr +   9758   9904  147  1  0  134   35   101 0.071   8.91
 1.04 Intr +  10302  10435  134  2  2    4   75    63 0.032  -4.88
 1.05 Intr +  12763  12979  217  1  1   97   84    78 0.265   5.88
 1.06 Intr +  15363  15421   59  2  2   95   46   106 0.088   3.86
 1.07 Intr +  18293  18483  191  2  2   39   56   127 0.037   2.91
 1.08 Term +  26161  26237   77  2  2   57   43   105 0.020  -0.08
 1.09 PlyA +  27474  27479    6                                1.05

 2.03 PlyA -  27633  27628    6                                1.05
```

Fig. 5.7. Genscan output: Header information



```
Predicted peptide sequence(s):

Predicted coding sequence(s):


>gi|GENSCAN_predicted_peptide_1|325_aa
MALISFTSPFNFIGKKSWQCITEAGFDKVDETIIFVISQSSRNVIVGEFLQDPCQGLPLL
KDLSSKQAANLFPWQRMEAVACDILLIMQPGHGQPAFLQGMSSRLSGAAEQVGSWSMRSQ
RHSLLWSVPEPVQQAGFLFPEALQSAGCFLPSNIGLQVLQFWTLGLTSVVCQGLSGLWPQ
IEGCTVGFSTFEVLGLGLASLLLSLQTAYCGTSPCDHSSSLSDSKAAVLENIGLLPLTHL
SECSRGGTQTGISGLKTELGAKVARVCQAEYGGESHAEREFWTPTEESLRVYKRGLISSA
SGISVDHGSLPEGLTKTFIPEGYEP

>gi|GENSCAN_predicted_CDS_1|978_bp
atggccctaatcagttttacatctccgtttaattttattggaaagaagagctggcaatgc
atcacagaggccggctttgacaaagtggatgaaacaattatcttcgttatcagccaaagc
agtagaaatgtgatagttggggaatttttgcaggacccatgccagggcttacctctgcta
aaggatttgtcctcaaagcaggcagcaaatctgttcccttggcagaggatggaagccgtg
gcttgtgacattctcctgataatgcagccaggccacgggcagccagcatttctgcagggg
atgagctccaggctcagtggggcagcagagcaagtggggagctggtccatgaggagtcag
cgtcattccttgctgtggtctgttcctgaaccagtccaacaggctggcttcctgttccca
gaagccctccaaagtgctggatgcttcctgccatcgaacattggactccaagttcttcag
ttttggactcttggacttacatcagtggtttgccagggactctcaggcctttggcctcag
attgaaggctgcactgtcggcttctctacttttgaggttttgggactcggactggcttcc
ttgctcctcagcttgcagacagcctattgtgggacttcaccttgtgatcattccagcagc
ctttcggattccaaagcggctgtcctggaaaatatagggctccttccactaacccacctc
tctgaatgcagcagaggtggaacccagacagggatcagtgggttaaagacagagctggga
gccaaggtagccagagtttgccaggcagagtatggcggagagagccacgcagagagagaa
ttctggacacctacggaggaatctcttcgagtatataaaagaggactgatcagcagtgca
```

Fig. 5.8. Genscan output: predicted sequences

Creating SwingGenscan

The `SwingGenScan` application is composed of four packages as described below:

- `org.jfb.genscan`: contains the Genscan API that provides a framework for a Genscan implementation. It makes the implementation more flexible by allowing us to optimize, thread, or queue requests and perform other manipulations without having to change the whole application; the way the implementation works is transparent to the application.
- `org.jfb.jgenscan`: a Genscan implementation of the framework defined by the `org.jfb.genscan` package.
- `org.jfb.util`: contains classes for performing operations such as extracting the peptide and genes from a Genscan prediction.
- `org.jfb.swinggenscan`: contains all the classes to build the `SwingGenScan` application.
- `GenScanResult`: contains the parsed peptide and the gene predictions.
- `ResultDialog`: a JDialog window that displays the result of Genscan operation. In this window, users can select one or more sequences to place into the BLAST pipeline using `SwingBlast`.
- `SwingGenScan`: the main application window where users can select the parameters for running a Genscan prediction against a chosen nucleotide sequence

The goal of this Chapter is to create a gene prediction and annotation pipeline which enables a user to perform gene prediction followed by further downstream analysis of the predicted gene and peptide sequences using BLAST. SwingGenScan uses SwingBlast to send Genscan predicted sequences for BLAST analysis. To enable this, we have modified `SwingBlast` version 2.5 that we created in Chapter 3 and separated the functionality provided by that application into four packages that we will use in `SwingGenScan`:

- `org.jfb.blast`: provides the BLAST API

- `org.jfb.jqblast`: provides an implementation of the BLAST API

- `org.jfb.util`: contains classes that provide functions that can be shared by more than one application (to enable future code reuse). For instance, the class QueryHelper in this package contains two methods (sendQuery and postQuery) to send GET or POST HTTP requests and the HTML result back.

- `org.jfb.swingblast3`: is the new refactored SwingBlast application. Since this is a major change, we have named this version 3.

The four classes can be packaged into a jar file called `swingblast.jar`. The jar file can serve as a library whose functionality can be used like any other Java library by placing it in the Java classpath. The structure of the `SwingGenScan` application is shown in **Fig. 5.9**.

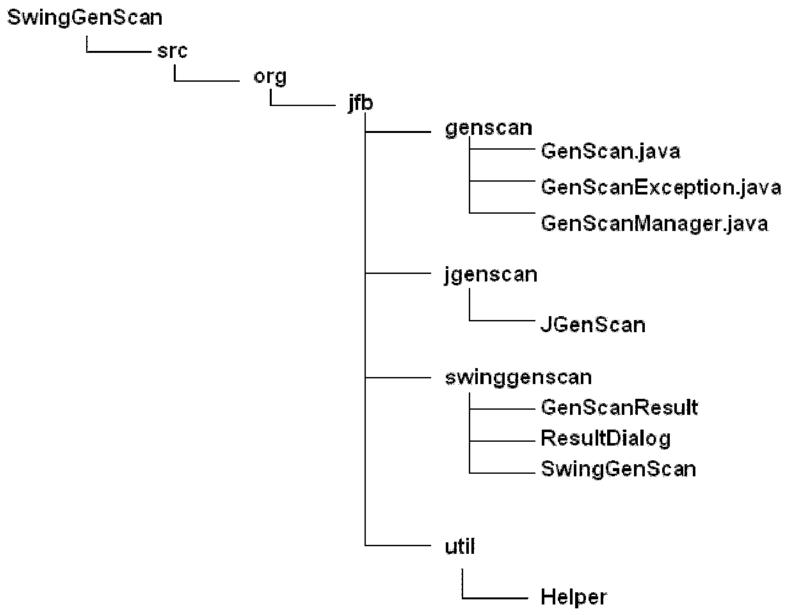


Fig. 5.9. The SwingGenScan application structure

Writing the Code for SwingGenScan

The `org.jfb.genscan` package contains the following Java classes:

`GenScan.java`

`GenScanException.java`, and

```
        GenScanManager.java
```

As described earlier, this package contains the API that provides a framework for a Genscan implementation. Let's look at the code for the first Java class `GenScan` located in the file `Genscan.java` (**Listing 5.1**).

Listing 5.1. Code for Java class GenScan

```
  package org.jfb.genscan;

  import java.util.HashMap;
  import java.util.Observable;

  public abstract class GenScan extends Observable {
      public   abstract   Object   submitQuery(Map   parameters)
throws GenScanException;

      public abstract Object requestResult(Object identifier)
            throws                       GenScanException,
IllegalArgumentException;
  }
```

When we run a Genscan analysis, we would like to know the status of the Genscan operation - has the request been submitted and if so, is the sequence currently in process, or has it encountered an error? The `Genscan` class provides a simple way of being notified of events through the use of the *observer pattern* as described in Chapter 2.

Next we define the GenScanManager class, whose purpose is to provide an instance of GenScan (**Listing 5.2**). As we'll see later, an implementation of the GenScan API will call the GenScanManager's register method to register itself as the default GenScan implementation.

Remember, we don't want to modify our code if we change the GenScan implementation to provide a multi-threaded, queued and multi-server implementation in the future. So to load our GenScan implementation we just pass the full name of the Java class to load, through the JVM system property (defined as "genscanClass.driver") using the –D option as explained earlier in Chapter 3. Another way is to call `Class.forName` ("full name of the Java class") to have the Java *classloader* locate the implementation and load it into the *JVM.* The reader will notice that the `createGenScan()` is *thread safe*, which means that a different instance of the Java `GenScan` implementation will be loaded for each thread and therefore it will not be a problem while accessing shared resources. For the same reason, multiple Genscan analyses can be run in a multi-threaded application. To return an instance of the implementation of GenScan we then use the *Java reflection API* (defined in *java.lang.reflect* package) to retrieve the constructor and create a new object of the GenScan implementation here called `JgenScan`.

Listing 5.2. GenScanManager.Java

```
  package org.jfb.genscan;

  public class GenScanManager {
      private static String genscanClass = null;
      private static boolean initialized = false;

      public   static   synchronized   void   register(GenScan
genscan) {
          genscanClass = genscan.getClass().getName();
          initialized = true;
      }

      private static void loadInitialDrivers() {
          final           String           driver           =
System.getProperty("genscanClass.driver");
          if (driver == null)
              return;

          try {
              System.out.println("GenScanManager.Initialize:
loading " + driver);
              Class.forName(driver);
          } catch (Exception e) {
              System.out.println("GenScanManager.Initialize:
load failed: " + e);
          }
      }

      public    static    GenScan    createGenScan()    throws
GenScanException {
          if (!initialized) {
              initialized = true;
              loadInitialDrivers();
          }
          if (genscanClass == null)
              throw new GenScanException("There is no driver
configured! "
                      + "Please use genscanClass.driver Java
property or Class.forName" +
                      " to load the driver class.");
          try {
              // In a multi thread environment we need to
//make sure that the class is loaded
              final    Class    aClass    =    (Class)
Class.forName(genscanClass, true,

Thread.currentThread().getContextClassLoader());
              return   (GenScan)   aClass.getConstructor(new
Class[]{}).newInstance(new Object[]{});
          } catch (Exception e) {
              throw new GenScanException(e);
          }
      }
  }
```

Next, we need to be able to get an instance of `GenScan`, or more specifically, an instance of the implementation that fulfills our Java `GenScan` declaration requirements. The design of the GenScan framework provided by the API we wrote is to make the implementation transparent to the user. For example, the implementation uses an HTTP server to run the Genscan analysis and to retrieve the result. This entire process is shielded from the user. The user simply calls the `submitQuery` method with a Map of parameters and requests a result using an object identifier.

The code below loads the class for the Genscan implementation:

```
  (Class aClass = (Class) Class.forName(genscanClass, true,

Thread.currentThread().getContextClassLoader());
              return   (GenScan)   aClass.getConstructor(new
Class[]{}).newInstance(new Object[]{});
```

We use *Java reflection* to retrieve a `Class` instance of the class defined by the name `genscanClass` by calling the static method `forName` from class `Class` and we cast it to `Class`. Then we use the `Class` instance we retrieved to construct an instance of that class by calling the `getConstructor` method that we cast also to type `GenScan`. *Casting* an object means forcing the object to be of a certain Java type. Of course, the type one wants to cast an object into must be one that the object inherits from. The new type can be an interface, an abstract class or a super class type. Casting is done in Java by putting the new type in parentheses before the object as shown above.

Note the static method in `GenScanManager.Java`:

```
public static synchronized void register(GenScan genscan) {
        genscanClass = genscan.getClass().getName();
        initialized = true;
    }
```

This method allows any implementation to register itself to the `GenScanManager` by calling it with an instance of an implementation of `GenScan` in a *static statement*. The method just stores the full Java class name of the implementation of `GenScan` by using Java reflection (`getClass()` method) on an object. The name will be then used by the `createGenScan()` method to provide an instance of `GenScan`.

Finally, the `GenScanException` class handles any exceptions that may arise during the operation of `Genscan` (**Listing 5.3**).

Listing 5.3. GenScanException class

```
package org.jfb.genscan;

public class GenScanException extends Exception {
    public GenScanException() {
    }

    public GenScanException(String message) {
        super(message);
    }

    public   GenScanException(String   message,   Throwable
cause) {
        super(message, cause);
    }

    public GenScanException(Throwable cause) {
        super(cause);
    }
}
```

Next we implement `Genscan` as shown in **Listing 5.4.** In the `JGenScan` class, the `register()` method is called by `createGenScan()` in case no Java class name for any implementation has been provided. Next the method `loadInitialDrivers()` will attempt to first retrieve the full Java class name of the implementation by looking at a JVM system property passed through the *JVM* as argument using the *–D option* as explained before:

```
java –DgenscanClass.driver=org.jfb.jgenscan.JgenScan
```

The line above will define in the system the property `genscanClass.driver` with the value `org.jfb.jgenscan.JgenScan`. We get the system property back in the Java code like this:

```
    System.getProperty("genscanClass.driver");
```

If the value found is not *null*, the method will then attempt to load the class through a class method call – `Class.forName()`. If `JGenScan` is not in the Java classpath, then the Java classloader will fail to load the class and will throw a `ClassNotFoundException`. So it is important to make sure that you declare `JGenScan` in the Java classpath. The method `forName()` has the effect of initializing the class implementing `GenScan`. Part of the initialization is to run the static statements and set up the static fields or constants.

Listing 5.4. The JGenScan class

```
  package org.jfb.jgenscan;

  import org.jfb.genscan.GenScan;
  import org.jfb.genscan.GenScanException;
  import org.jfb.genscan.GenScanManager;
  import org.jfb.util.QueryHelper;

  import java.io.UnsupportedEncodingException;
  import java.net.URLEncoder;
  import java.util.ArrayList;
  import java.util.Collection;
  import java.util.HashMap;

  public class JGenScan extends GenScan {
      private   static   final   String   GENSCAN_HOSTNAME   =
"genes.mit.edu";
      private   static   final   String   GENSCAN_PATH   =   "/cgi-
bin/oldgenscanw.cgi";
      private static final int GENSCAN_PORT = 80;
      private static final String GENSCAN_URL = "http://" +
GENSCAN_HOSTNAME + ":" + GENSCAN_PORT + "/" + GENSCAN_PATH;

      static {
          System.out.println("Registering        "        +
JGenScan.class);
          GenScanManager.register(new JGenScan());
      }


      private   static   Map   reqIdToResultFileName   =   new
HashMap();
      private   Collection   currentRunningGenScan   =   new
ArrayList();
      private static final int NUMBER_OF_SECOND = 3000;

      public   Object   submitQuery(Map   parameters)   throws
GenScanException {
          final         String         urlapiQuery         =
createUrlapiQuery(parameters);
          setChanged();
          notifyObservers("Submitting the job to the server
with query\n" + urlapiQuery);
          Runnable runnable = new Runnable() {
              public void run() {
                  Object res;
                  try {
                      res                                  =
QueryHelper.sendQuery(urlapiQuery, GENSCAN_URL, true);
                  } catch (Throwable e) {
                      res   =   new   GenScanException("Problem
with URL " + GENSCAN_URL, e);
```

```
                  }
                  final String key = "" + this.hashCode();
                  synchronized (reqIdToResultFileName) {
                      System.out.println("Storing the result
...");
                      reqIdToResultFileName.put(key, res);
                  }
              }
          };
          new Thread(runnable).start();
          final String key = "" + runnable.hashCode();
          currentRunningGenScan.add(key);
          return key;
      }

      public Object requestResult(Object identifier) throws
GenScanException {
          if (!currentRunningGenScan.contains(identifier))
              throw new IllegalArgumentException(identifier +
" has no corresponding result!");
          Map tmp = null;
          boolean hasFinished = false;
          int ct = 0;
          synchronized (this) {
              while (!hasFinished) {
                  tmp = new HashMap(reqIdToResultFileName);
                  hasFinished = tmp.containsKey(identifier);
                  if (hasFinished) {

reqIdToResultFileName.remove(identifier);
                      break;
                  }
                  setChanged();
                  notifyObservers("Waiting        "        +
NUMBER_OF_SECOND
                          + " seconds before re-trying (total
waiting time: "
                          +   (ct   +=   NUMBER_OF_SECOND)   +
"s).");
                  try {
                      wait(NUMBER_OF_SECOND);
                  } catch (InterruptedException e) {
                      e.printStackTrace();
                  }
              }
          }
          final Object o = tmp.get(identifier);
          if (o == null) return null;
          if (o instanceof Throwable)
              throw       new       GenScanException("Embedded
exception", (Throwable) o);
          return o;
      }
      private String createUrlapiQuery(Map parameters) {
          StringBuffer query = new StringBuffer();
          try {
              final Object org = parameters.get("organism");
              final Object nam = parameters.get("name");
              final          Object          sub          =
parameters.get("subOptExonCutoff");
              final          Object          dis          =
parameters.get("displayOption");
              query.append("-
s=").append(URLEncoder.encode((String)
parameters.get("sequence"), "UTF-8"));
              if (org != null) {
                  query.append("&-
o=").append(URLEncoder.encode((String) org, "UTF-8"));
              }
              if (nam != null) {
                  query.append("&-
n=").append(URLEncoder.encode((String) nam, "UTF-8"));
              }
              if (sub != null) {
                  query.append("&-
e=").append(URLEncoder.encode((String) sub, "UTF-8"));
              }
              if (dis != null) {
                  query.append("&-
p=").append(URLEncoder.encode((String) dis, "UTF-8"));
              }
          } catch (UnsupportedEncodingException uee) {
              uee.printStackTrace();
          }
          return query.toString();
      }
  }
```

Note the following piece of code in **Listing 5.4**:

```
          new Thread(runnable).start();
          final String key = "" + runnable.hashCode();
          currentRunningGenScan.add(key);
          return key;
```

Here, we are threading the process to be able to run more than one query without having to wait for the first one to finish. Also because we're running in *a* multi-threaded environment we want to synchronize the Map called `reqIdToResultFileName`, to safely save the right key with the right result and to avoid more than one thread to modify the Map at the same time that could potentially populate the Map with wrong data.

After we have submitted the query, we retrieve the result by calling the `requestResult()` method. That method will return only when the result is available. One has to make sure that a call to that method is not executed in the event-dispatching thread, because that will block the repaint of the application.

The method `requestResult()` described in **Listing 5.4** first checks that the request identifier is a valid argument. If invalid, the method will throw an exception that would allow us to track down multiple calls to the method with the same argument that could probably imply an infinite loop. We are protecting multiple threads from accessing the same block when we are checking if the request is ready, by surrounding the block with a *synchronized ()* block. The synchronization is on the current object "*this*" calling that method. That means that the JVM will set a lock (a unique token) on the current object to the thread that first entered the block. Then, until the thread inside that block releases the lock, any other threads waiting to run that piece of code will have to wait for the lock to be released. The actual processes are transparent to the developer because of the use of the *synchronized* Java keyword.

The result of the `Genscan` operation is stored in the `GenScanResult` object. This is essentially the predicted peptide and gene sequences and any additional data about the search that the user may wish to save such as the name of the server, the Genscan parameters used for the prediction as well as the time taken to execute the prediction etc. The code for the `GenScanResult` class is shown in **Listing 5.5**.

Listing 5.5. GenScanResult.Java

```
package org.jfb.swinggenscan;

public class GenScanResult {
    private String[] peptideGene = null;

    public void setPeptideAndGene(String[] pepGene) {
        peptideGene = pepGene;
    }

    public String[] getPeptideGene() {
        return peptideGene;
    }
}
```

Next, the `ResultDialog class` takes a `GenScanResult` object and displays its content.

```
    public void showResult(GenScanResult result) {
        String[] pepGene = result.getPeptideGene();
        if (pepGene == null) {
            list.setCellRenderer(new
DefaultListCellRenderer());
            list.setListData(new    String[]{"No    Results
Found"});
        } else {
            list.setCellRenderer(new MyListCellRenderer());
            list.setListData(pepGene);
        }
    }
```

The `ResultDialog class` also allows the user to run additional analyses to be run on the predicted gene and peptide sequences. In this case, we will add a functionality to perform a BLAST search on user selected Genscan predictions. To do that, we add a check box against each predicted sequence and a button called "`Run Blast`" at the bottom. Once the user selects a sequence and hits the "`Run BLAST`" button, the `SwingBlast` application we created earlier is invoked with the selected sequences in the text area of the `SwingBlast` application.

```
        runBlastButton = new JButton("Run Blast");
        runBlastButton.addActionListener(new
ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (!list.isSelectionEmpty()) {

SwingBlast3.launch(list.getSelectedValues()[0].toString());
```

```
                }
```

The code for the `ResultDialog` class is shown in **Listing 5.6.**

Listing 5.6. ResultDialog.java

```
  package org.jfb.swinggenscan;

  import org.jfb.swingblast3.SwingBlast3;
  import javax.swing.*;
  import javax.swing.event.ListSelectionEvent;
  import javax.swing.event.ListSelectionListener;
  import java.awt.*;
  import java.awt.event.ActionEvent;
  import java.awt.event.ActionListener;

  public class ResultDialog extends JDialog {
      private   static   final   Dimension   BD_PREF_SIZE   =   new
Dimension(530, 460);
      private JList list;
      private JButton runBlastButton;

      public      ResultDialog(Frame      owner)      throws
HeadlessException {
          super(owner);
          setTitle("GenScan Result Dialog");
      }

      public void init() {
          list = new JList();

list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
          list.addListSelectionListener(new
ListSelectionListener() {
              public  void  valueChanged(ListSelectionEvent  e)
{
                  if (!e.getValueIsAdjusting()) {

runBlastButton.setEnabled(!list.isSelectionEmpty());
                  }
              }
          });
          JScrollPane scrollPaneArea = new JScrollPane(list);
          scrollPaneArea.setPreferredSize(new  Dimension(500,
400));
          JPanel panel = new JPanel();
          panel.setLayout(new BorderLayout());
          panel.add(scrollPaneArea, BorderLayout.NORTH);

          JPanel buttonPane = new JPanel();
          buttonPane.setLayout(new      BoxLayout(buttonPane,
BoxLayout.LINE_AXIS));
          buttonPane.add(Box.createHorizontalGlue());
          buttonPane.add(Box.createRigidArea(new
Dimension(10, 0)));

          runBlastButton = new JButton("Run Blast");
          runBlastButton.addActionListener(new
ActionListener() {
              public void actionPerformed(ActionEvent e) {
                  if (!list.isSelectionEmpty()) {

SwingBlast3.launch(list.getSelectedValues()[0].toString());
                  }
              }
          });
          runBlastButton.setSize(new Dimension(80, 20));
          runBlastButton.setEnabled(false);
          buttonPane.add(runBlastButton);
          panel.add(runBlastButton, BorderLayout.SOUTH);
          getContentPane().add(panel);
          setSize(BD_PREF_SIZE);
          setVisible(true);
      }

      public void showResult(GenScanResult result) {
          String[] pepGene = result.getPeptideGene();
          if (pepGene == null) {
              list.setCellRenderer(new
DefaultListCellRenderer());
              list.setListData(new    String[]{"No    Results
Found"});
          } else {
              list.setCellRenderer(new MyListCellRenderer());
              list.setListData(pepGene);
          }
      }


      private  static  class  MyListCellRenderer  implements
ListCellRenderer {
          public Component getListCellRendererComponent(JList
list,  final  Object  value,  int  index,  boolean  isSelected,
boolean cellHasFocus) {
              JPanel jPanel = new JPanel();
              jPanel.setLayout(new BorderLayout());
              final     JTextArea     textArea     =     new
JTextArea(value.toString());
              final Font sf = textArea.getFont();
              Font f = new Font("Monospaced", sf.getStyle(),
sf.getSize());
              textArea.setFont(f);
              textArea.setLineWrap(true);
              final JCheckBox comp = new JCheckBox();
```

```
                comp.setSelected(isSelected);
                jPanel.add(comp, BorderLayout.WEST);
                jPanel.add(textArea, BorderLayout.CENTER);
                return jPanel;
            }
        }
    }
```

The SwingGenScan User Interface

The application interface is created using swing libraries. **Listing 5.7** shows the code for the `SwingGenScan` application.

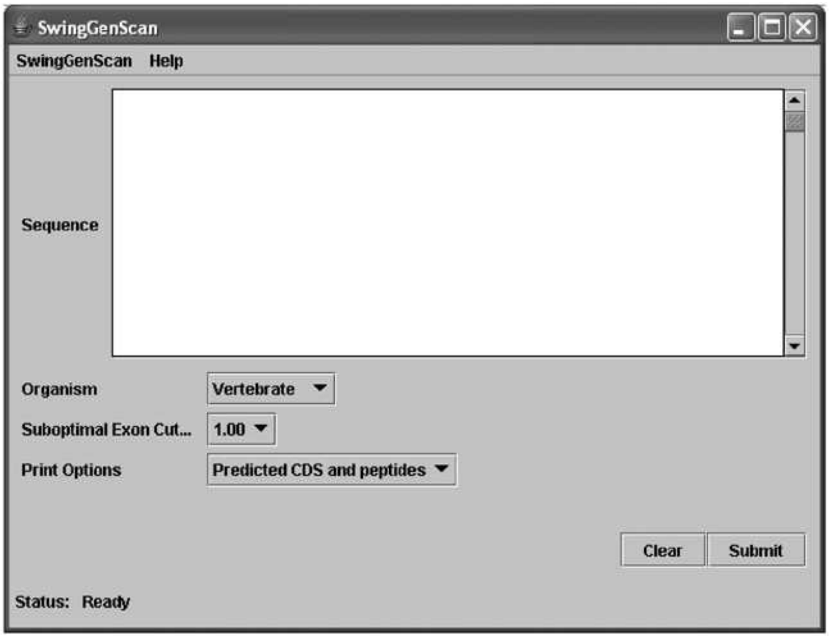
Listing 5.7. SwingGenScan user interface

```
    package org.jfb.swinggenscan;


    import org.jfb.genscan.GenScan;
    import org.jfb.genscan.GenScanException;
    import org.jfb.genscan.GenScanManager;
    import org.jfb.util.Helper;

    import javax.swing.*;
    import javax.swing.event.DocumentEvent;
    import javax.swing.event.DocumentListener;
    import java.awt.*;
    import java.awt.event.ActionEvent;
    import java.awt.event.ActionListener;
    import java.util.HashMap;
    import java.util.Observable;
    import java.util.Observer;

    public class SwingGenScan extends JFrame {
        private static final String APP_NAME = "SwingGenScan";
        private  static  final  String  APP_VERSION  =  "Version
1.0";
        private static final String STATUS_LABEL = "Status: ";
        private static final String STATUS_READY = "Ready";

        private  static  final  Dimension  LABEL_PREFERRED_SIZE  =
new Dimension(127, 16);
        private  static  final  Dimension  COMBO_PREFERRED_SIZE  =
new Dimension(60, 25);
        private  static  final  Dimension  CP_PREF_SIZE  =  new
Dimension(450, 410);

        private static final String[] ORGANISMS =
                new    String[]{"Vertebrate",    "Arabidopsis",
"Maize"};
      private static final String[] PRINT_OPTIONS =
              new    String[]{"Predicted    peptides    only",
"Predicted CDS and peptides"};
      private          static          final          String[]
SUBOPTIMAL_EXON_CUTOFF_VALUES =
              new  String[]{"1.00",  "0.50",  "0.25",  "0.10",
"0.05", "0.02", "0.01"};

      private JComponent newContentPane;
      private JTextArea sequenceArea;
      private JScrollPane scrollPaneArea;
      private JLabel statusLabel;
      private JLabel statusText;

      private JComboBox organisms;
      private JComboBox printOptions;
      private JComboBox exonCutoffs;

      private JButton clearBtn, submitBtn;

      private JMenuItem aboutItem;
      private JMenuItem quitItem;

      public SwingGenScan() {
          super();
          seqFormInit();
      }

      private void seqFormInit() {
          setTitle(APP_NAME);
          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          newContentPane = new JPanel();
          newContentPane.setOpaque(true);
          newContentPane.setLayout(new BorderLayout());

   getContentPane().add(newContentPane)
          setContentPane(newContentPane);

          // Create the menu bar
          JMenuBar menu = new JMenuBar();
          JMenu swingBlastMenu = new JMenu(APP_NAME);
          quitItem = new JMenuItem("Quit");
          swingBlastMenu.add(quitItem);
          menu.add(swingBlastMenu);

          JMenu helpMenu = new JMenu("Help");
          aboutItem = new JMenuItem("About");
          helpMenu.add(aboutItem);
          menu.add(helpMenu);
          setJMenuBar(menu);
          // Create the sequence pane
          JPanel sequencePanel = new JPanel();
          JLabel sequence = new JLabel("Sequence");
          sequenceArea = new JTextArea();
          final Font sf = sequenceArea.getFont();
          Font  f  =  new  Font("Monospaced",  sf.getStyle(),
sf.getSize());
          sequenceArea.setFont(f);
          sequenceArea.setLineWrap(true);
          scrollPaneArea = new JScrollPane(sequenceArea);
          scrollPaneArea.setPreferredSize(new  Dimension(300,
200));

          sequencePanel.setLayout(new
BoxLayout(sequencePanel, BoxLayout.LINE_AXIS));
          sequencePanel.add(sequence);
          sequencePanel.add(Box.createRigidArea(new
Dimension(10, 0)));
          sequencePanel.add(scrollPaneArea);

sequencePanel.setBorder(BorderFactory.createEmptyBorder(10,
0, 10, 0));

          statusLabel = new JLabel(STATUS_LABEL);
          statusLabel.setPreferredSize(new      Dimension(50,
30));
          statusText = new JLabel(STATUS_READY);
          JPanel statusPanel = new JPanel();

statusPanel.setBorder(BorderFactory.createEmptyBorder(0,   5,
5, 5));
          statusPanel.setLayout(new BorderLayout());
          statusPanel.add(statusLabel, BorderLayout.WEST);
          statusPanel.add(statusText, BorderLayout.CENTER);

          // Lay out the buttons from left to right

          JPanel buttonPane = new JPanel();
          submitBtn = new JButton("Submit");
          clearBtn = new JButton("Clear");

          buttonPane.setLayout(new      BoxLayout(buttonPane,
BoxLayout.LINE_AXIS));
          buttonPane.add(Box.createHorizontalGlue());
          buttonPane.add(Box.createRigidArea(new
Dimension(10, 0)));
          buttonPane.add(clearBtn);
          buttonPane.add(submitBtn);

          JPanel jPanel = new JPanel();
          jPanel.setLayout(new BorderLayout());
          jPanel.setBorder(BorderFactory.createEmptyBorder(0,
10, 10, 10));
```

```
          jPanel.add(sequencePanel, BorderLayout.NORTH);
          jPanel.add(createProgramPanel(),
BorderLayout.CENTER);
          jPanel.add(buttonPane, BorderLayout.SOUTH);

          newContentPane.add(jPanel, BorderLayout.CENTER);
          newContentPane.add(statusPanel,
BorderLayout.SOUTH);
          newContentPane.setPreferredSize(CP_PREF_SIZE);
          enableFunctions(false);
          // Display the window
          pack();
          Dimension                screenSize                 =
Toolkit.getDefaultToolkit().getScreenSize();
          setLocation((screenSize.width - CP_PREF_SIZE.width)
/ 2,
                  (screenSize.height - CP_PREF_SIZE.height) /
2);
          setVisible(true);
          addListeners();
      }

      private JPanel createProgramPanel() {
          JPanel organismPanel = new JPanel();
          JLabel organismLabel = new JLabel("Organism");

organismLabel.setPreferredSize(LABEL_PREFERRED_SIZE);
          organisms = new JComboBox(ORGANISMS);
          organisms.setMaximumSize(COMBO_PREFERRED_SIZE);
          organismPanel.setLayout(new
BoxLayout(organismPanel, BoxLayout.LINE_AXIS));
          organismPanel.add(organismLabel);
          organismPanel.add(Box.createRigidArea(new
Dimension(10, 0)));
          organismPanel.add(organisms);
          organismPanel.add(Box.createRigidArea(new
Dimension(5, 0)));
          organismPanel.add(Box.createHorizontalGlue());

          JPanel exonCutoffPanel = new JPanel();
          JLabel   exonCutoffLabel   =   new   JLabel("Suboptimal
Exon Cuttoff");

exonCutoffLabel.setPreferredSize(LABEL_PREFERRED_SIZE);
          exonCutoffs                  =                   new
JComboBox(SUBOPTIMAL_EXON_CUTOFF_VALUES);
          exonCutoffs.setMaximumSize(COMBO_PREFERRED_SIZE);
          exonCutoffPanel.setLayout(new
BoxLayout(exonCutoffPanel, BoxLayout.LINE_AXIS));
          exonCutoffPanel.add(exonCutoffLabel);
          exonCutoffPanel.add(Box.createRigidArea(new
Dimension(10, 0)));
          exonCutoffPanel.add(exonCutoffs);
          exonCutoffPanel.add(Box.createRigidArea(new
Dimension(5, 0)));
          exonCutoffPanel.add(Box.createHorizontalGlue());

          JPanel printOptionsPanel = new JPanel();
          JLabel   printOptionsLabel   =   new   JLabel("Print
Options");

printOptionsLabel.setPreferredSize(LABEL_PREFERRED_SIZE);
          printOptions = new JComboBox(PRINT_OPTIONS);
          printOptions.setMaximumSize(COMBO_PREFERRED_SIZE);
          printOptionsPanel.setLayout(new
BoxLayout(printOptionsPanel, BoxLayout.LINE_AXIS));
          printOptionsPanel.add(printOptionsLabel);
          printOptionsPanel.add(Box.createRigidArea(new
Dimension(10, 0)));
          printOptionsPanel.add(printOptions);
          printOptionsPanel.add(Box.createRigidArea(new
Dimension(5, 0)));
          printOptionsPanel.add(Box.createHorizontalGlue());

          JPanel paramPanel = new JPanel();
          paramPanel.setLayout(new      BoxLayout(paramPanel,
BoxLayout.PAGE_AXIS));

          paramPanel.add(organismPanel);
          paramPanel.add(Box.createRigidArea(new Dimension(0,
5)));
          paramPanel.add(exonCutoffPanel);
          paramPanel.add(Box.createRigidArea(new Dimension(0,
5)));
          paramPanel.add(printOptionsPanel);
          paramPanel.add(Box.createRigidArea(new Dimension(0,
5)));

          return paramPanel;
      }

      private void addListeners() {
          quitItem.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent e) {
                  System.exit(0);
              }
          });

          aboutItem.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent e) {

JOptionPane.showMessageDialog(org.jfb.swinggenscan.SwingGenSc
an.this, APP_NAME + " " + APP_VERSION,
                              "About      "      +      APP_NAME,
JOptionPane.INFORMATION_MESSAGE);
              }
          });

          clearBtn.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent e) {
                  sequenceArea.setText("");
                  enableFunctions(false);
                  statusText.setText(STATUS_READY);
              }
          });

          submitBtn.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent e) {

                  Runnable runnable = new Runnable() {
                      public void run() {
                          GenScan genScan = null;
                          try {

Class.forName("org.jfb.jgenscan.JGenScan");
                              genScan                =
GenScanManager.createGenScan();
                          }   catch   (ClassNotFoundException
cnfe) {
                              cnfe.printStackTrace();
                          } catch (GenScanException gse) {
                              gse.printStackTrace();
                          }
                          Map param = new HashMap();
                          param.put("sequence",
sequenceArea.getText());
                          param.put("organism",
organisms.getSelectedItem());
                          param.put("subOptExonCutoff",
exonCutoffs.getSelectedItem());
                          param.put("displayOption",
printOptions.getSelectedItem());
                          Object requestIdentifier = null;
                          try {
                              requestIdentifier          =
genScan.submitQuery(param);
                          } catch (GenScanException gse) {
                              gse.printStackTrace();
                          }
                          Observer observer = new Observer()
{
                              public  void  update(Observable
o, Object arg) {

SwingGenScan.this.statusText.setText(arg.toString());
                              }
                          };
                          genScan.addObserver(observer);
                          Object text = null;
                          try {
                              text                          =
genScan.requestResult(requestIdentifier);
                          } catch (GenScanException gse) {
                              gse.printStackTrace();
                          }
                          final   GenScanResult   result    =
Helper.extractPeptideAndGene(text.toString());
                          EventQueue.invokeLater(new
Runnable() {
                              public void run() {

statusText.setText(STATUS_READY);
                                  final          ResultDialog
resultDialog = new ResultDialog(SwingGenScan.this);
                                  resultDialog.init();

resultDialog.showResult(result);
                              }
                          });
                      }
                  };
                  new Thread(runnable).start();
              }
          });

          sequenceArea.getDocument().addDocumentListener(new
DocumentListener() {
              public void insertUpdate(DocumentEvent e) {

enableFunctions(sequenceArea.getText().trim().length() > 0);
              }

              public void removeUpdate(DocumentEvent e) {

enableFunctions(sequenceArea.getText().trim().length() > 0);
              }

              public void changedUpdate(DocumentEvent e) {
              }
          });
      }


      private void enableFunctions(boolean enabled) {
          organisms.setEnabled(enabled);
          exonCutoffs.setEnabled(enabled);
          printOptions.setEnabled(enabled);
      }

      public static void main(String[] args) {
          SwingUtilities.invokeLater(new Runnable() {
              public void run() {
```

```
                        new SwingGenScan();
                    }
                });
            }
        }
```

The `SwingGenScan` user interface is shown in **Fig. 5.10**.



Sequence

Organism Vertebrate

Suboptimal Exon Cut... 1.00

Print Options Predicted CDS and peptides

Status: Ready

Fig. 5.10. SwingGenScan user interface

After the Genscan prediction has finished, we need to parse the raw results, which are presented as an HTML page to extract the actual predicted gene and peptide sequences. This is done through the `Helper` class within the `org.jfb.util` package. We have created a separate package for this to enable developers to use this code in a different application that requires similar functionality without the need to extract it from the main application or block of code (**Listing 5.8**).

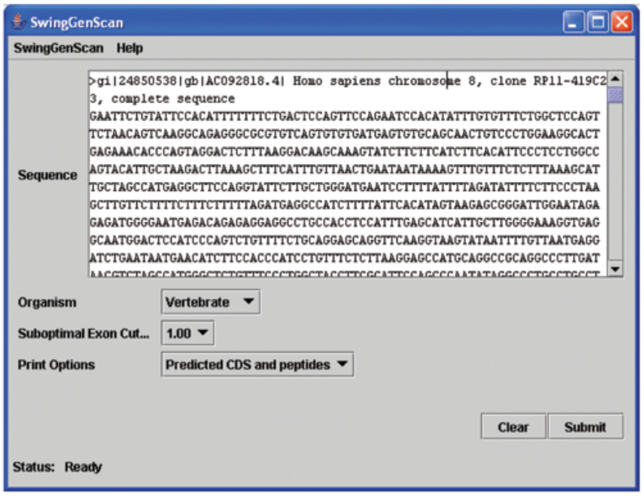
Listing 5.8. `org.jfb.util package`

```
package org.jfb.util;

import org.jfb.swinggenscan.GenScanResult;
```

```
  import java.util.ArrayList;
  import java.util.Collection;

  public class Helper {
      public                static                GenScanResult
extractPeptideAndGene(String rawHtml) {
          final    String    begin    =    "Predicted    peptide
sequence(s):";
          final String end = "<b>Explanation</b>";
          String                allSequences                 =
rawHtml.substring(rawHtml.indexOf(begin)   +   begin.length(),
rawHtml.indexOf(end));
          if (allSequences.indexOf("NO PEPTIDES PREDICTED") >
0) {
              return new GenScanResult();
          }
          int beginIndex = allSequences.indexOf('>');
          allSequences  =  allSequences.substring(beginIndex  +
1, allSequences.length());
          beginIndex = allSequences.indexOf('>');
          allSequences   =   allSequences.substring(beginIndex,
allSequences.length());
          final String[] results = allSequences.split("\n");
          Collection sequences = new ArrayList();
          StringBuffer sb = new StringBuffer();
          for (int i = 0; i < results.length; i++) {
              final String line = results[i];
              if (line.trim().length() == 0) {
                  sequences.add(sb.toString());
                  sb = new StringBuffer();
              } else {
                  sb.append(line).append("\n");
              }
          }
          sequences.add(sb.toString());
          String[] res = new String[sequences.size()];
          sequences.toArray(res);
          final GenScanResult result = new GenScanResult();
          result.setPeptideAndGene(res);
          return result;
      }
  }
```

Running SwingGenScan

Fig. 5.11 to **Fig. 5.14** demonstrate a typical run of the `SwingGenScan` application beginning with the pasting of a sequence – in this case – the complete sequence of the human chromosome number 8 (GI number 24850538) and the printing of the predicted genes and peptides.



```
>gi|24850538|gb|AC092818.4| Homo sapiens chromosome 8, clone RP11-419C2
3, complete sequence
GAATTCTGTATTCCACATTTTTTTCTGACTCCAGTTCCAGAATCCACATATTTGTGTTTCTGGCTCCAGT
TCTAACAGTCAAGGCAGAGGGCGCGTGTCAGTGTGTGATGAGTGTGCAGCAACTGTCCCTGGAAGGCACT
GAGAAACACCCAGTAGGACTCTTTAAGGACAAGCAAAGTATCTTCTTCATCTTCACATTCCCTCCTGGCC
AGTACATTGCTAAGACTTAAAGCTTTCATTTGTTAACTGAATAATAAAAGTTTGTTTCTCTTTAAAGCAT
TGCTAGCCATGAGGCTTCCAGGTATTCTTGCTGGGATGAATCCTTTTATTTTAGATATTTTCTTCCCTAA
GCTTGTTCTTTTCTTTCTTTTTAGATGAGGCCATCTTTTATTCACATAGTAAGAGCGGGATTGGAATAGA
GAGATGGGGAATGAGACAGAGAGGAGGCCTGCCACCTCCATTTGAGCATCATTGCTTGGGGAAAGGTGAG
GCAATGGACTCCATCCCAGTCTGTTTTCTGCAGGAGCAGGTTCAAGGTAAGTATAATTTTGTTAATGAGG
ATCTGAATAATGAACATCTTCCACCCATCCTGTTTCTCTTAAGGAGCCATGCAGGCCGCAGGCCCTTGAT
```

Sequence

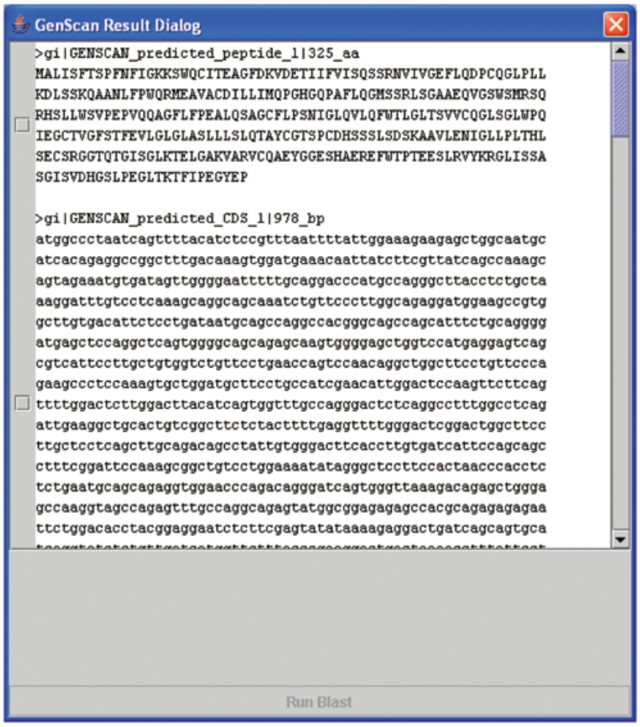
Organism: Vertebrate

Suboptimal Exon Cut...: 1.00

Print Options: Predicted CDS and peptides

Status: Ready

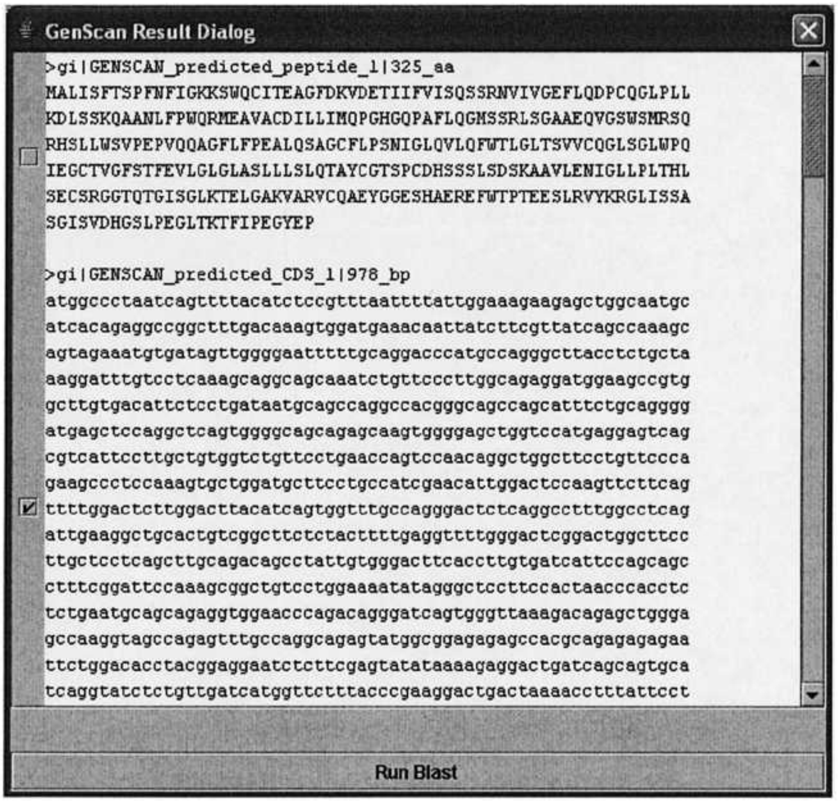
Fig. 5.11. Running SwingGenScan



```
>gi|GENSCAN_predicted_peptide_1|325_aa
MALISFTSPFNFIGKKSWQCITEAGFDKVDETIIFVISQSSRNVIVGEFLQDPCQGLPLL
KDLSSKQAANLFPWQRMEAVACDILLIMQPGHGQPAFLQGMSSRLSGAAEQVGSWSMRSQ
RHSLLWSVPEPVQQAGFLFPEALQSAGCFLPSNIGLQVLQFWTLGLTSVVCQGLSGLWPQ
IEGCTVGFSTFEVLGLGLASLLLSLQTAYCGTSPCDHSSSLSDSKAAVLENIGLLPLTHL
SECSRGGTQTGISGLKTELGAKVARVCQAEYGGESHAEREFWTPTEESLRVYKRGLISSA
SGISVDHGSLPEGLTKTFIPEGYEP

>gi|GENSCAN_predicted_CDS_1|978_bp
atggccctaatcagttttacatctccgtttaattttattggaaagaagagctggcaatgc
atcacagaggccggctttgacaaagtggatgaaacaattatcttcgttatcagccaaagc
agtagaaatgtgatagttggggaattttgcaggacccatgccagggcttacctctgcta
aaggatttgtcctcaaagcaggcagcaaatctgttcccttggcagaggatggaagccgtg
gcttgtgacattctcctgataatgcagccaggccacgggcagccagcatttctgcagggg
atgagctccaggctcagtggggcagcagagcaagtggggagctggtccatgaggagtcag
cgtcattccttgctgtggtctgttcctgaaccagtccaacaggctggcttcctgttccca
gaagccctccaaagtgctggatgcttcctgccatcgaacattggactccaagttcttcag
ttttggactcttggacttacatcagtggtttgccagggactctcaggcctttggcctcag
attgaaggctgcactgtcggcttctctacttttgaggttttgggactcggactggcttcc
ttgctcctcagcttgcagacagcctattgtgggacttcaccttgtgatcattccagcagc
ctttcggattccaaagcggctgtcctggaaaatatagggctccttccactaacccacctc
tctgaatgcagcagaggtggaacccagacagggatcagtgggttaaagacagagctggga
gccaaggtagccagagtttgccaggcagagtatggcggagagagccacgcagagagagaa
ttctggacacctacggaggaatctcttcgagtatataaaagaggactgatcagcagtgca
```

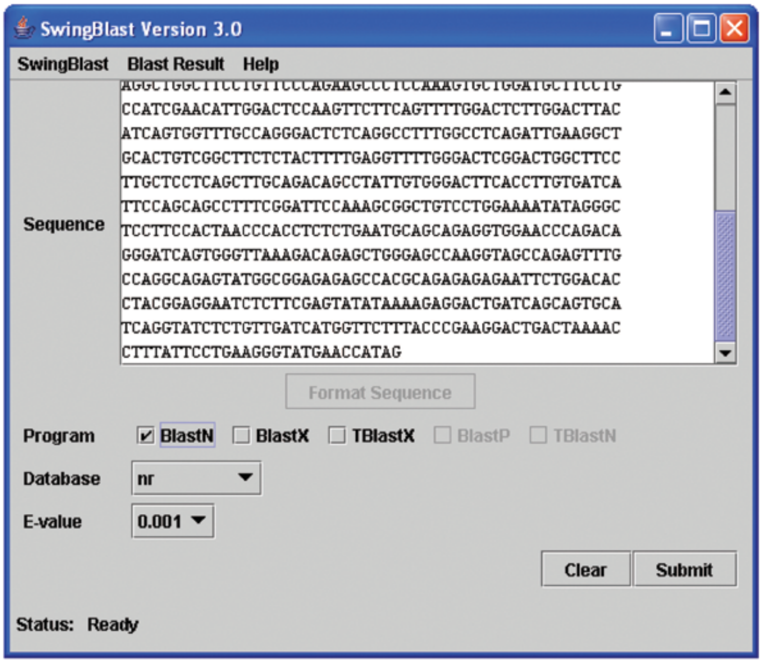
Fig. 5.12. SwingGenScan results

The "`Run Blast`" button remains disabled as long as no sequence is selected for BLAST analysis and becomes active after a sequence is selected (**Fig. 5.12** and **Fig. 5.13**). **Fig. 5.13** and **Fig. 5.14** further demonstrate how predicted sequences can be selected and sent for further analysis using BLAST. Note that selected sequences can be unselected by simultaneously pressing the Control and the left click button on the Mouse (on Windows) and the Apple button and the click (on Mac).



```
>gi|GENSCAN_predicted_peptide_1|325_aa
MALISFTSPFNFIGKKSWQCITEAGFDKVDETIIFVISQSSRNVIVGEFLQDPCQGLPLL
KDLSSKQAANLFPWQRMEAVACDILLIMQPGHGQPAFLQGMSSRLSGAAEQVGSWSMRSQ
RHSLLWSVPEPVQQAGFLFPEALQSAGCFLPSNIGLQVLQFWTLGLTSVVCQGLSGLWPQ
IEGCTVGFSTFEVLGLGLASLLLSLQTAYCGTSPCDHSSSLSDSKAAVLENIGLLPLTHL
SECSRGGTQTGISGLKTELGAKVARVCQAEYGGESHAERE FWTPTEESLRVYKRGLISSA
SGISVDHGSLPEGLTKTFIPEGYEP

>gi|GENSCAN_predicted_CDS_1|978_bp
atggccctaatcagttttacatctccgtttaattttattggaaagaagagctggcaatgc
atcacagaggccggctttgacaaagtggatgaaacaattatcttcgttatcagccaaagc
agtagaaatgtgatagttggggaatttttgcaggacccatgccagggcttacctctgcta
aaggatttgtcctcaaagcaggcagcaaatctgttcccttggcagaggatggaagccgtg
gcttgtgacattctcctgataatgcagccaggccacgggcagccagcatttctgcagggg
atgagctccaggctcagtggggcagcagagcaagtggggagctggtccatgaggagtcag
cgtcattccttgctgtggtctgttcctgaaccagtccaacaggctggcttcctgttccca
gaagccctccaaagtgctggatgcttcctgccatcgaacattggactccaagttcttcag
ttttggactcttggacttacatcagtggtttgccagggactctcaggcctttggcctcag
attgaaggctgcactgtcggcttctctacttttgaggttttgggactcggactggcttcc
ttgctcctcagcttgcagacagcctattgtgggacttcaccttgtgatcattccagcagc
ctttcggattccaaagcggctgtcctggaaaatatagggctccttccactaacccacctc
tctgaatgcagcagaggtggaacccagacagggatcagtgggttaaagacagagctggga
gccaaggtagccagagtttgccaggcagagtatggcggagagagccacgcagagagagaa
ttctggacacctacggaggaatctcttcgagtatataaaagaggactgatcagcagtgca
tcaggtatctctgttgatcatggttctttacccgaaggactgactaaaacctttattcct
```

Fig. 5.13. Selecting sequences for BLAST analysis



Sequence

```
AGGCTGGCTTCCTGTTCCCAGAAGCCCTCCAAAGTGCTGGATGCTTCCTG
CCATCGAACATTGGACTCCAAGTTCTTCAGTTTTGGACTCTTGGACTTAC
ATCAGTGGTTTGCCAGGGACTCTCAGGCCTTTGGCCTCAGATTGAAGGCT
GCACTGTCGGCTTCTCTACTTTTGAGGTTTTGGGACTCGGACTGGCTTCC
TTGCTCCTCAGCTTGCAGACAGCCTATTGTGGGACTTCACCTTGTGATCA
TTCCAGCAGCCTTTCGGATTCCAAAGCGGCTGTCCTGGAAAATATAGGGC
TCCTTCCACTAACCCACCTCTCTGAATGCAGCAGAGGTGGAACCCAGACA
GGGATCAGTGGGTTAAAGACAGAGCTGGGAGCCAAGGTAGCCAGAGTTTG
CCAGGCAGAGTATGGCGGAGAGAGCCACGCAGAGAGAGAATTCTGGACAC
CTACGGAGGAATCTCTTCGAGTATATAAAAGAGGACTGATCAGCAGTGCA
TCAGGTATCTCTGTTGATCATGGTTCTTTACCCGAAGGACTGACTAAAAC
CTTTATTCCTGAAGGGTATGAACCATAG
```

Program [✓ BlastN] BlastX TBlastX BlastP TBlastN

Database nr

E-value 0.001

Status: Ready

Fig. 5.14. Sending predicted genes to SwingBlast for BLAST analysis

Only BLASTN has been implemented in the `SwingGenScan` application for the purpose of demonstration. The user can further develop the application by adding functionality for other BLAST operations. The Genscan-BLAST analysis pipeline can be implemented in a completely different manner than described here. For example, the `Genscan` output window displaying the gene and peptide predictions can be modified to contain the appropriate widgets to perform multiple BLAST analyses on multiple selected sequences without the intermediate step of invoking the `SwingBlast` application. The implementation shown here is one of many ways to achieve the same end-result.

Summary

In this Chapter, we have demonstrated how we can create a basic gene prediction and annotation pipeline by connecting the Genscan and BLAST programs together. We created the BLAST application separately and tied it together with Genscan thereby building an analytic pipeline that demonstrates reuse of existing code libraries. The addition of functionality to Genscan to enable BLAST analysis of predicted sequences is an example of a real-life use case that will have much practical utility for researchers who are involved in the sequencing and study of new genomes.

Questions and Exercises

1. The `SwingGenScan` application created in the Chapter demonstrated the ability to perform BLASTN searches. Extend the application to enable other types of BLAST searches (BLASTX, BLASTP, etc.).

2. An important goal of gene prediction is to decipher gene structure – that is, the location of exons and introns – in the input nucleotide sequence. Think about how you would identify intron-exon boundaries from Genscan predictions and align the individual introns and exons along the original nucleotide sequence.

Additional Resources

- GenomeScan - http://genes.mit.edu/genomescan.html
- Glimmer - http://www.cbcb.umd.edu/software/glimmer/
- HMMGene - http://www.cbs.dtu.dk/services/HMMgene/
- TwinScan - http://genes.cs.wustl.edu/

Selected Reading

Prediction of complete gene structures in human genomic DNA. Burge, C. and Karlin, S. (1997) J. Mol. Biol. 268, 78-94.

Finding the genes in genomic DNA. Burge, C. B. and Karlin, S. (1998) Curr. Opin. Struct. Biol. 8, 346-354.

Computational inference of homologous gene structures in the human genome. Yeh, R.-F., Lim, L. P., and Burge, C. B. (2001) Genome Res. 11: 803-816.

Improved microbial gene identification with GLIMMER (1999) A.L. Delcher, D. Harmon, S. Kasif, O. White, and S.L. Salzberg. Nucleic Acids Research 27:23, 4636-4641.

Two methods for improving performance of an HMM and their application for gene finding. In Proc. of Fifth Int. Conf. on Intelligent Systems for Molecular Biology, ed. Gaasterland, T. et al., Menlo Park, CA: AAAI Press, 1997, pp. 179-186.