Geneious 9.1
Phobos Plugin Development Example

Dr Shane Sturrock - Biomatters Ltd

February 29, 2016

Introduction

This document is not intended to teach Java. It does assume that you have a programming background but if you want to know more about the specifics of Java then Sun's Tutorial (`http://java.sun.com/docs/books/tutorial/java/index.html`) is a good place to start. Knowledge of Eclipse is also useful but not essential.

Starting with the `ExampleSequenceAnnotationGenerator`, we will walk through the development of a Geneious 9.1 plugin wrapper for a command line application.

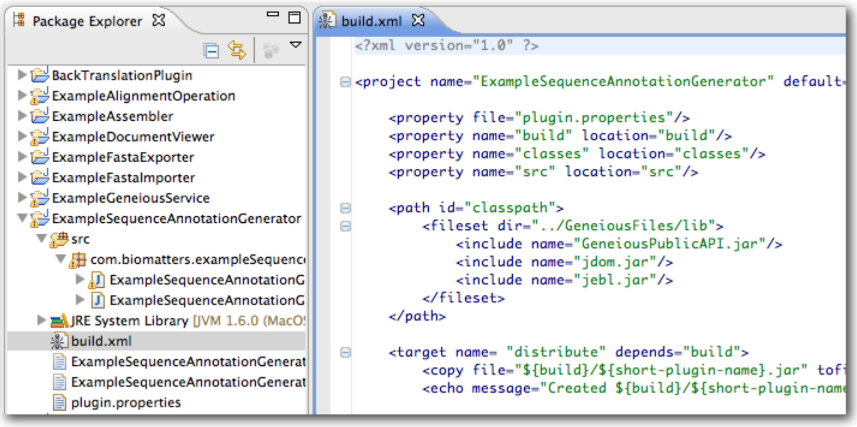
The `ExampleSequenceAnnotationGenerator` example project is pre-configured for Eclipse and you can use the refactoring feature to rename the classes to match the name of your project. Java is case-sensitive so follow the instructions exactly.

Before starting, download and install the latest version of Geneious 9.1 from the Geneious website (`http://www.geneious.com`). Open Geneious and make sure you uninstall the 'Phobos Tandem Repeat Finder plugin' if it is installed. You will be replacing this with your own version while learning how such a plugin is developed in stages.

Setting up your project

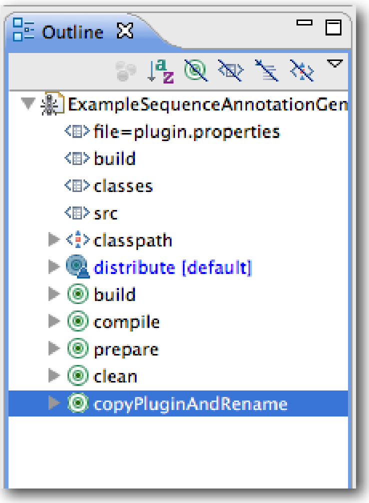
Make sure you have followed the instructions in `setup.html` that came with your downloaded copy of the Plugin Development Kit. This will have taken you through the process of setting up Eclipse with the example projects since you're going to base this new plugin on one of those.

Rather than edit the `ExampleSequenceAnnotationGenerator` itself, the `build.xml` file has a target that can duplicate and rename the project. Open the `build.xml` file by double clicking on it in the Package Explorer of Eclipse.

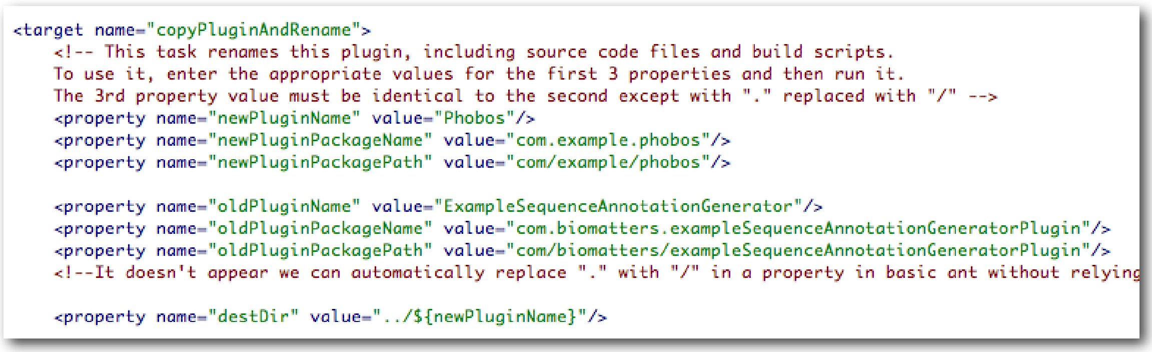

Package Explorer
BackTranslationPlugin
ExampleAlignmentOperation
ExampleAssembler
ExampleDocumentViewer
ExampleFastaExporter
ExampleFastaImporter
ExampleGeneiousService
ExampleSequenceAnnotationGenerator
src
com.biomatters.exampleSequenc
ExampleSequenceAnnotationG
ExampleSequenceAnnotationG
JRE System Library [JVM 1.6.0 (MacO
build.xml
ExampleSequenceAnnotationGenerat
ExampleSequenceAnnotationGenerat
plugin.properties
build.xml

```
<?xml version="1.0" ?>

<project name="ExampleSequenceAnnotationGenerator" default

    <property file="plugin.properties"/>
    <property name="build" location="build"/>
    <property name="classes" location="classes"/>
    <property name="src" location="src"/>

    <path id="classpath">
        <fileset dir="../GeneiousFiles/lib">
            <include name="GeneiousPublicAPI.jar"/>
            <include name="jdom.jar"/>
            <include name="jebl.jar"/>
        </fileset>
    </path>

    <target name= "distribute" depends="build">
        <copy file="${build}/${short-plugin-name}.jar" tof
        <echo message="Created ${build}/${short-plugin-name
```

In the 'Outline' pane you should see the `ExampleSequenceAnnotationGenerator` which you can expand and select the `copyPluginAndRename` target and the editor will jump to the right position in the `build.xml` file.

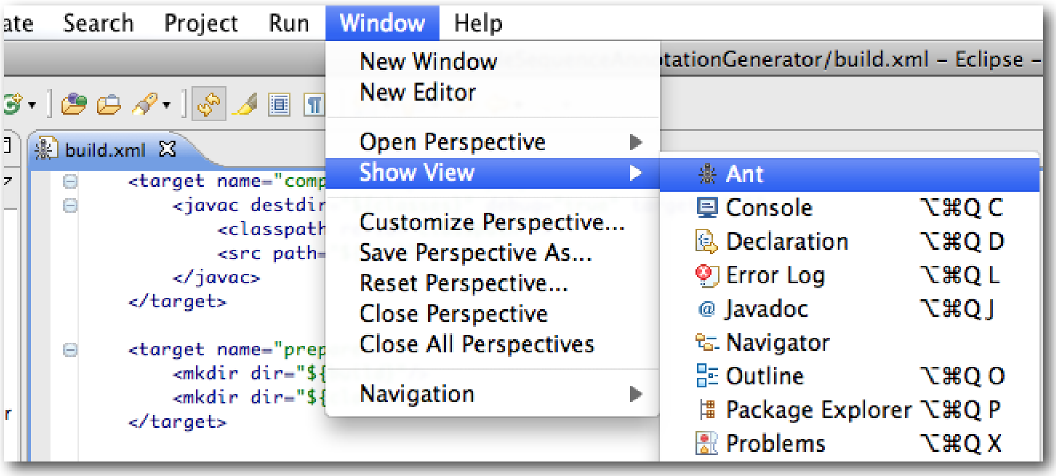


Change the `newPluginName`, `newPluginPackageName` and `newPluginPackagePath` as shown below.



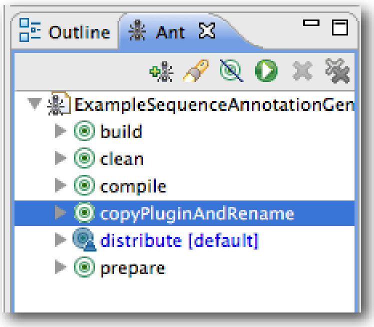
```
<target name="copyPluginAndRename">
    <!-- This task renames this plugin, including source code files and build scripts.
    To use it, enter the appropriate values for the first 3 properties and then run it.
    The 3rd property value must be identical to the second except with "." replaced with "/" -->
    <property name="newPluginName" value="Phobos"/>
    <property name="newPluginPackageName" value="com.example.phobos"/>
    <property name="newPluginPackagePath" value="com/example/phobos"/>

    <property name="oldPluginName" value="ExampleSequenceAnnotationGenerator"/>
    <property name="oldPluginPackageName" value="com.biomatters.exampleSequenceAnnotationGeneratorPlugin"/>
    <property name="oldPluginPackagePath" value="com/biomatters/exampleSequenceAnnotationGeneratorPlugin"/>
    <!--It doesn't appear we can automatically replace "." with "/" in a property in basic ant without relying

    <property name="destDir" value="../${newPluginName}"/>
```

Save the changes to `build.xml` and then open the **Ant** view. and drag the `build.xml` file from `ExampleSequenceAnnotationGenerator` into it and expand it.



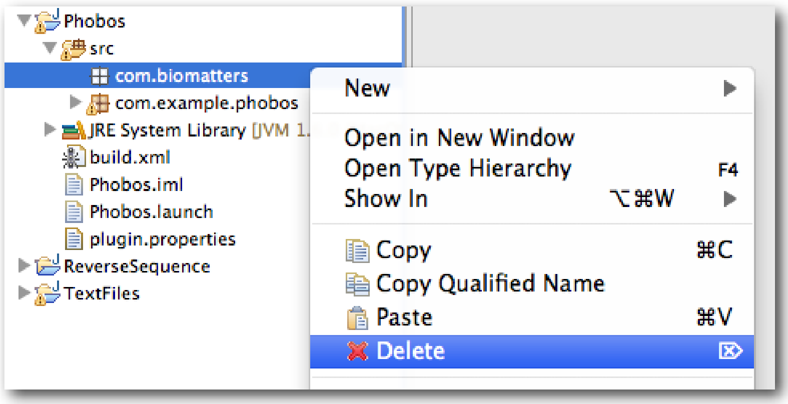
```
    <target name="comp
        <javac destdir
            <classpath
            <src path=
        </javac>
    </target>

    <target name="prep
        <mkdir dir="${
        <mkdir dir="${
    </target>
```

Now double click on the `copyPluginAndRename` target and a new Phobos plugin will be created in the same `examples` directory as the others.

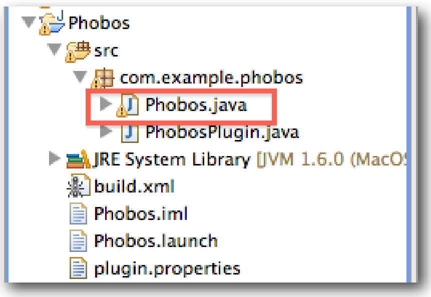


You will now need to import it into Eclipse using **File** → **Import** and select **General** → **Existing Projects into Workspace** and click 'Next', browse to the Geneious API `examples` folder and you should now see a `Phobos` folder so select that and click 'Open'. Don't copy the project into your workspace, just click 'Finish'. You can now close the `build.xml` file you have open and collapse the `ExampleSequenceAnnotationGenerator` package before expanding out the new `Phobos` package you just imported.

As part of the project copying you'll see a `com.biomatters` package which you won't be using so you can just delete it.



You also need to remove the `Phobos.java` class since this won't be used.



You're now ready to start working with the code.

PhobosPlugin.java

Open the `PhobosPlugin.java` file by double clicking on its name in the Package Explorer.

The refactoring has already renamed the package and class correctly so you're ready to start writing the code. All plugins you develop are implementations of the `GeneiousPlugin` abstract class.

```
package com.example.phobos;

import com.biomatters.geneious.publicapi.plugin.GeneiousPlugin;

public class PhobosPlugin extends GeneiousPlugin {
```

There are seven compulsory methods already present and these allow you to provide information that identifies the plugin.

Since you're developing this plugin, it is time to put your stamp on it and set the authorship. Modify the `getAuthors()` method to return your name where I've put mine.

```
public String getAuthors() {
  return "Shane Sturrock";
}
```

Next modify the `getDescription()` method to return a simple description of what the plugin does. Since Phobos is a DNA-satellite search tool that is a suitable description.

```
public String getDescription() {
  return "DNA-satellite search tool";
}
```

The next method returns help. The Eclipse IDE will identify this line as unresolved for the moment. We'll take advantage of the features of the IDE to quickly create the class and populate it with the right methods shortly.

```
public String getHelp() {
  return PhobosAnnotationGenerator.HELP;
}
```

Give the plugin a descriptive name.

```
public String getName() {
  return "Phobos Tandem Repeat Finder";
}
```

Lastly, set the version number so users can identify if they have the latest version of your plugin. When you start developing you might like to start with 0.1 and keep incrementing the digit. You can go past 0.9 without being required to go to 1.0 so 0.12 and so on are perfectly acceptable. You can also have three digits if you like so in this case, the first version will be called 0.0.1 as shown. Do not use non-numeric values such as letters or punctuation.

```
public String getVersion() {
  return "0.0.1";
}
```

At this point, you have provided all of the necessary information for your plugin except for the part which actually does something useful.

Phobos is a command line application which produces a list of repeats that it found for a given sequence. What your plugin needs to do is take a sequence from a Geneious document, pass that to Phobos along with the appropriate command line parameters, and then parse the results back and mark the identified repeats on the original sequence document.

A `SequenceAnnotationGenerator` generates annotations on a sequence, set of sequences (sequence list), or an alignment document so we will override `GeneiousPlugin`'s `getSequenceAnnotationGenerators()` method.

Modify the code to match the following:

```
public class PhobosPlugin extends GeneiousPlugin  {
  public SequenceAnnotationGenerator[] getSequenceAnnotationGenerators() {
    return new SequenceAnnotationGenerator[] {
      new PhobosAnnotationGenerator()
    };
  }
```

You now have two lines with errors, both referring to a currently non-existent class. Mouse over the first error and select 'Create class', and it will create the `PhobosAnnotationGenerator` class. Now the only error should be related to 'HELP' which we'll resolve in the next section.

You have now finished with `PhobosPlugin.java` for the moment.

PhobosAnnotationGenerator.java

This class is where all of the work is done. Open the file in your Eclipse editor and you should see the following including the predefined methods.

```
package com.example.phobos;

import com.biomatters.geneious.publicapi.plugin.DocumentSelectionSignature;
import com.biomatters.geneious.publicapi.plugin.GeneiousActionOptions;
import com.biomatters.geneious.publicapi.plugin.SequenceAnnotationGenerator;

class PhobosAnnotationGenerator extends SequenceAnnotationGenerator {

  public GeneiousActionOptions getActionOptions() {
    return null;
  }

  public String getHelp() {
    return null;
  }

  public DocumentSelectionSignature[] getSelectionSignatures() {
    return null;
  }
}
```

The first thing to do is deal with the error which Eclipse is flagging from `PhobosPlugin.java` which indicates `PhoboAnnotationGenerator.HELP` cannot be resolved. At the top of the class just create a static final string called 'HELP'.

```
class PhobosAnnotationGenerator extends SequenceAnnotationGenerator {

  static final String HELP = "Phobos detects tandem repeats in DNA sequence(s)";
```

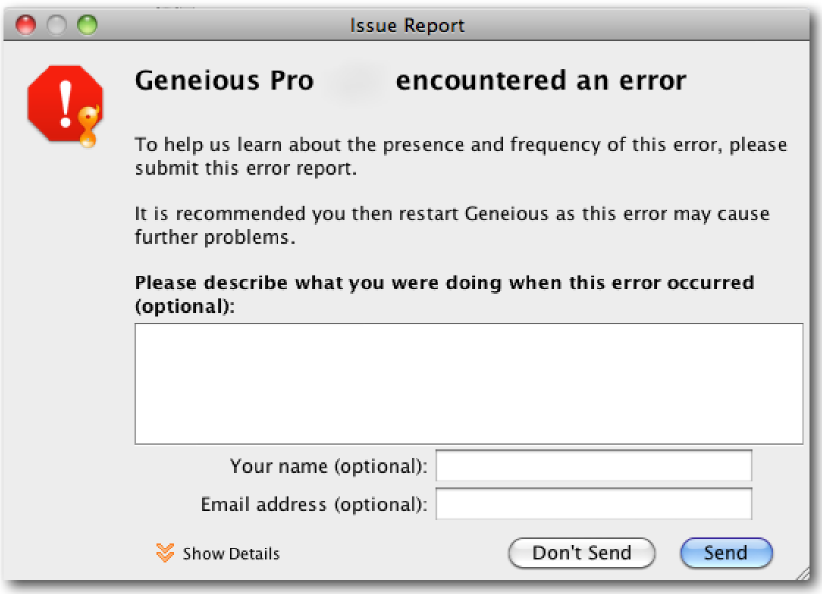
Next edit the `getHelp()` method to return this new string.

```
public String getHelp() {
  return HELP;
}
```

If you save `PhobosAnnotationGenerator.java` now, the error indication on `PhobosPlugin.java` will go away.

At this point, it should be possible to get your plugin to compile without errors. To do this, press the **debug** button in the Eclipse tool bar and this will launch Geneious with your plugin installed.

Geneious will complain and you are going to see this type of message often so you might as well see it now.



Issue Report

Geneious Pro encountered an error

To help us learn about the presence and frequency of this error, please submit this error report.

It is recommended you then restart Geneious as this error may cause further problems.

Please describe what you were doing when this error occurred (optional):

Your name (optional):

Email address (optional):

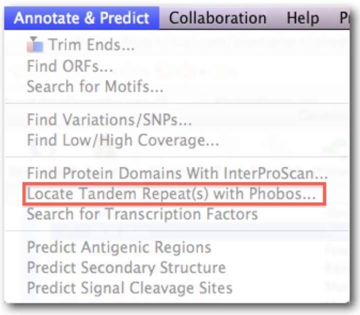
If you're interested, you can click the 'Show Details' button and you will see that the error was a `NullPointerException`. This isn't surprising because your plugin isn't finished. Click the 'Don't Send' button and open **Geneious Preferences** → **Plugins and Features**. Click the 'Customize Feature Set...' button and scroll down the list to find the Phobos plugin. Click on the 'Info' button and you'll find the description and so on that you specified in the `PhobosPlugin` class.

Quit Geneious again and we'll make the plugin actually do something.

First Geneious needs to know where in the various menus it should put your new operation. In `PhobosAnnotationGenerator` edit the `GeneiousActionOptions` as follows:

```
public GeneiousActionOptions getActionOptions() {
  // Put the menu item in the Find/Locate part of the Sequence menu
  return new GeneiousActionOptions("Locate Tandem Repeat(s) with Phobos...")
                .setMainMenuLocation(GeneiousActionOptions.MainMenu.AnnotateAndPredict);
}
```

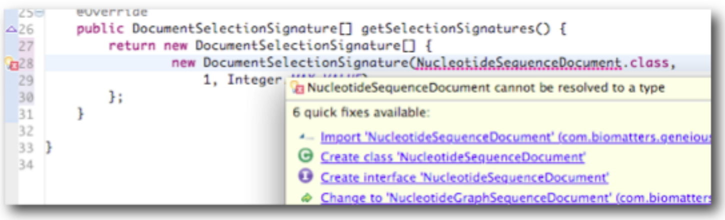
This will add the menu item to the '`Annotate & Predict`' menu. By default, this will be at the bottom of the menu. Save and then start Geneious with the **Ant** button to see this new menu item in the '`Annotate & Predict`' menu.


Annotate & Predict
Collaboration
Help
Trim Ends...
Find ORFs...
Search for Motifs...
Find Variations/SNPs...
Find Low/High Coverage...
Find Protein Domains With InterProScan...
Locate Tandem Repeat(s) with Phobos...
Search for Transcription Factors
Predict Antigenic Regions
Predict Secondary Structure
Predict Signal Cleavage Sites

Note that the new item is greyed out regardless of what document you select. This is because you need to add a selection signature which identifies which documents your plugin will work with. Modify `DocumentSelectionSignature[]` to match the next section of code.

```
public DocumentSelectionSignature[] getSelectionSignatures() {
  return new DocumentSelectionSignature[] {
                    new DocumentSelectionSignature(NucleotideSequenceDocument.class,
                           1, Integer.MAX_VALUE)
  };
}
```

Eclipse will probably complain that `NucleotideSequenceDocument` cannot be resolved to a type. It will offer a number of fixes and in this case the correct fix is to import '`NucleotideSequenceDocument`' which will get rid of the error.



```
    @Override
    public DocumentSelectionSignature[] getSelectionSignatures() {
        return new DocumentSelectionSignature[] {
                new DocumentSelectionSignature(NucleotideSequenceDocument.class,
                    1, Integer
        };
    }

}
```

NucleotideSequenceDocument cannot be resolved to a type

6 quick fixes available:

Import 'NucleotideSequenceDocument' (com.biomatters.geneiou
Create class 'NucleotideSequenceDocument'
Create interface 'NucleotideSequenceDocument'
Change to 'NucleotideGraphSequenceDocument' (com.biomatters

We want Phobos to work on nucleotide sequence documents as single sequences, groups of selected sequences and sequence lists (including sequence alignments). `Integer.MAX_VALUE` is a predefined constant specifying the largest value an `int` can hold ($2^{31} - 1 = 2147483647$). It is highly unlikely that you would use anything like this value and it might make sense to set the value to something lower like 1,000 depending on how long it would take to perform the operation but for now setting it to this maximum possible value is fine.

You can now test that selecting nucleotide documents of various types enables your new menu item and non-nucleotide documents do not. If you try and activate the menu Geneious will still fail because the `SequenceAnnotationGenerator generate()` method hasn't been implemented so let's do that now. Note Eclipse may catch the exception so check in the debug perspective.

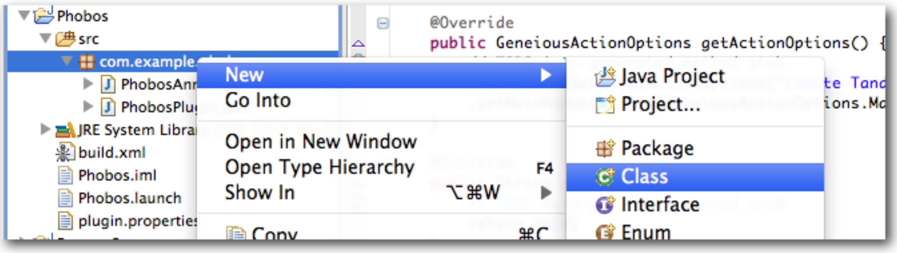
```
@Override
public List<SequenceAnnotationGenerator.AnnotationGeneratorResult>
generate(AnnotatedPluginDocument[] documents,
              SequenceAnnotationGenerator.SelectionRange selectionRange,
              ProgressListener progressListener, Options options)
              throws DocumentOperationException {

       List<SequenceAnnotationGenerator.AnnotationGeneratorResult> resultsList =
              new ArrayList<SequenceAnnotationGenerator.AnnotationGeneratorResult>();

       // Code will go in here

       return resultsList;
}
```

This method generates a list of annotated sequences and will be the one which actually executes the `phobos` binary. There are some interesting points to note here. The method takes in an `AnnotatedPluginDocument[]`array which will be acted upon. There are other parameters which are needed but can be ignored for the moment such as `progressListener` and `options` although we will be using them later. We are not going to use `selectionRange` but it still needs to be present. As before, add imports as suggested to remove the errors for `AnnotatedPluginDocument`, `ProgressListener` (use the `jebl` version), `Options` (use the Biomatters version), `DocumentOperationException` and `ArrayList` and `List` (use the `java.util` ones for both otherwise you'll see errors).

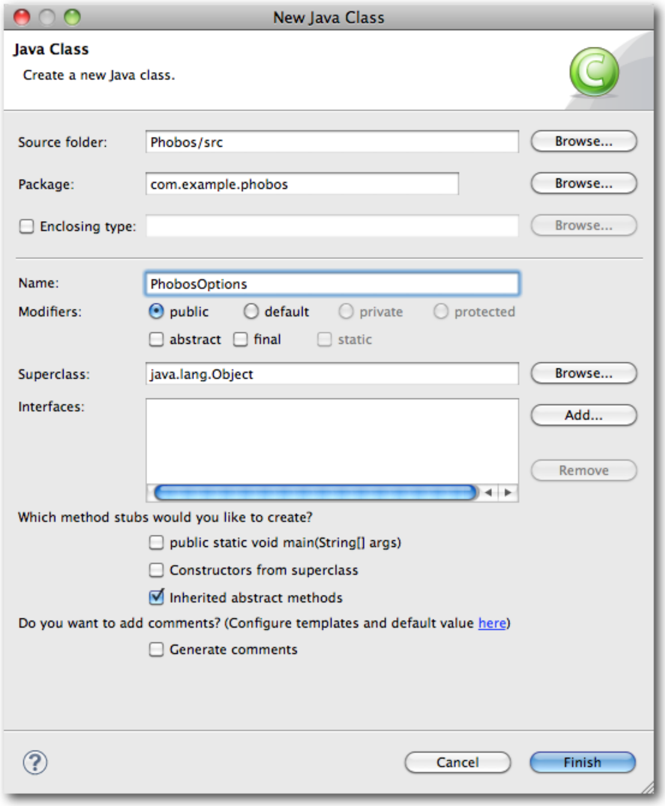
PhobosOptions.java

In order to tell Geneious where the `phobos` binary is there needs to be a window asking the user where it is. This is implemented by extending the `Options` class.

You need to create a new class called PhobosOptions.java so right click on the package name (`com.example.phobos`) and select **New** → **Class**.



This will bring up a dialogue where you can enter the name of the new class (`PhobosOptions`) and click 'Finish'.


New Java Class

Java Class

Create a new Java class.

Source folder: Phobos/src

Package: com.example.phobos

☐ Enclosing type:

Name: PhobosOptions

Modifiers: [✓ public] default private protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

Open the `PhobosOptions.java` file and change the code to match the following:

```
public class PhobosOptions extends Options {

  public PhobosOptions() {
    codeLocationOptions();
  }

  private FileSelectionOption codeLocation;

  private void codeLocationOptions() {
    codeLocation = addFileSelectionOption("EXE", "The Phobos executable:", "");
    codeLocation.setRestoreDefaultApplies(false);
  }
}
```

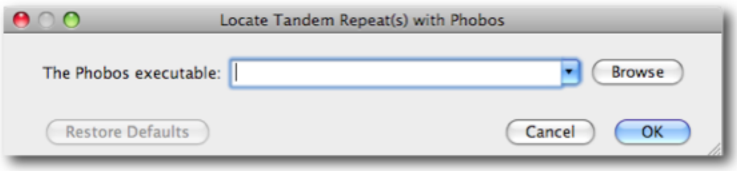
You'll need to add the import for Biomatters' `Options`.

You also need to add the following to `PhobosAnnotationGenerator.java`. Anywhere within the class will do but after the `DocumentSelectionSignature[]` method is a good place.

```
public Options getOptions(AnnotatedPluginDocument[] documents,
              SequenceAnnotationGenerator.SelectionRange selectionRange)
throws DocumentOperationException {
       return new PhobosOptions();
}
```

If you run now you should be able to select one or more nucleotide documents and select the '`Locate Tandem Repeat(s) with Phobos...`' menu item and Geneious will pop up an options window requesting the location of the phobos binary as seen here. Use the **Browse** button to locate the binary. If you haven't already downloaded Phobos, you can get it from the developer's website by pointing your web browser at "`http://www.rub.de/spezzoo/cm/cm_phobos.htm`" and selecting the correct version for your computer.

This option is set to ignore the **Restore Defaults** button since you don't want to have to keep locating the executable if you reset other parameters you will add later.



For the moment, that is all we will do with `PhobosOptions` since this is enough information for Geneious to run the executable. Later, we will add the various options to control its behaviour.

Running Phobos

Returning to `PhobosAnnotationGenerator.java`, replace the comment "`// Code will go here`" with the following which will loop through all of the sequence documents:

```
List<SequenceAnnotationGenerator.AnnotationGeneratorResults> resultsList =
       new ArrayList<SequenceAnnotationGenerator.AnnotationGeneratorResult>();

PhobosOptions phobosOptions = (PhobosOptions) options;

// temporary file to store FASTA sequence
File tempFile = null;

for (AnnotatedPluginDocument annotatedPluginDocument:documents) {

  SequenceDocument seqDoc =
          (SequenceDocument) annotatedPluginDocument.getDocument();

  tempFile = writeFasta(tempFile, seqDoc);

  String[] command = phobosOptions.getCommand(tempFile.getAbsolutePath());

}

return resultsList;
```

Phobos takes a FASTA formatted input file so for each sequence we will need to export a FASTA file from Geneious which phobos can then read. Since the `writeFasta()` and `getCommand()` don't exist yet, Eclipse will flag the errors. Also, notice the cast of options to `PhobosOptions` allowing access to the `phobosOption.getCommand()` method (defined on the next page.)

Each `SequenceDocument` includes the sequence and additional information such as the name and description which are needed for FASTA format which is very basic as shown here:

```
>name description
ATGGTAGAGTGA
```

This makes it very simple to write out each sequence in this format so now we'll implement the `writeFasta()` method inside the the `PhobosAnnotationGenerator` class:

```
private File writeFasta(File tempFile, SequenceDocument seqDoc)
throws DocumentOperationException {
  // Create temporary FASTA file for Phobos to read.
  try {
    // creates a temp file which will be deleted next time Geneious starts.
    tempFile = FileUtilities.createTempFile("temp_phobos", ".fasta", false);

    // write to FASTA format
    BufferedWriter out = new BufferedWriter(new FileWriter(tempFile));
    out.write(">"+seqDoc.getName()+" "+seqDoc.getDescription()+"\n");
    out.write(seqDoc.getSequenceString().toUpperCase());
    out.close();
  } catch (IOException e) {
    throw new DocumentOperationException(
           "Failed to write FASTA file:" + e.getMessage(), e);
  }
  return tempFile;
}
```

This code will create a temporary file and write the FASTA format sequence for use by Phobos. As usual, there will be some missing imports so add them to get rid of the warnings.

The next step is to build up the command line. The simple way to do this is to create an `ArrayList` and then add each command. This can then be converted into a `String[]` array which is what the `Execution` class requires. Add the following method to the `PhobosOptions` class:

```
// Return the command line options
String[] getCommand(String tempFile) {
  ArrayList<String> commandList = new ArrayList<String>();

  commandList.add(codeLocation.getValue());
  commandList.add(tempFile);

  // Make sure standard Phobos output is used
  commandList.add("--outputFormat 0");
  // This will be the command used by execute
  return commandList.toArray(new String[commandList.size()]);
}
```

Notice that this method requires the location of `tempFile` which is going to be passed in from `PhobosAnnotationGenerator` after it writes the FASTA file. We also need to ensure that Phobos uses the standard output format since we don't want any surprises when we try to parse the output.

Now it gets interesting because we have to run Phobos and collect the results to map back onto the sequence as annotations. Add the following code inside the `for` loop in `PhobosAnnotationGenerator.java` after the `getCommand` line:

```
String[] command = phobosOptions.getCommand(tempFile.getAbsolutePath());

try {
  // Need a listener on the output stream
  PhobosExecutionOutputListener outputListener =
    new PhobosExecutionOutputListener();
  // Build the Execution object with the command line, listeners and
  // nucleic Acid Sequence
  Execution exec = new Execution(command,progressListener,outputListener,(String)null,false);
  // Run the command
  exec.execute();
  // If it returns having been killed there was a problem
  // and the exception will be trapped by the 'catch'
  if (exec.wasKilledByGeneious()) {
    return null;
  }
  resultsList.add(outputListener.getResults());
} catch (IOException e) {
    throw new DocumentOperationException("Phobos failed:" + e.getMessage(), e);
} catch (InterruptedException e) {
    throw new DocumentOperationException("Process Killed!", e);
}
```

Note that Eclipse is warning you about `PhobosExecutionOutputListener` which doesn't yet exist. Mouse over the error and Eclipse will offer to create the class. Do so and we'll move on and fix up any issues there.

PhobosExecutionOutputListener.java

This class will read from `stdout` and `stderr` because Phobos writes its output to those streams. The class will have to capture those streams and parse the results.

The first step is to create somewhere for the annotations to be stored so they can be returned by this class as well as implement the `getResults()` method which will do the actual returning.

Modify `PhobosExecutionOutputListener` to match the following with any required imports and unimplemented methods:

```
class PhobosExecutionOutputListener extends Execution.OutputListener {

  final SequenceAnnotationGenerator.AnnotationGeneratorResult results =
      new SequenceAnnotationGenerator.AnnotationGeneratorResult();

  SequenceAnnotationGenerator.AnnotationGeneratorResult getResults() {
    return results;
  }
```

The next step is to modify the auto-generated `stderrWritten()` and `stdoutWritten()` method stubs to do something. Simply writing the streams to `stdout` is a good idea when debugging since running Geneious from within the Eclipse Debug perspective will allow you to see any errors that Phobos is returning.

```
@Override
public void stderrWritten(String output) {
  // Pass error messages through flagging them as coming from stderr
  System.out.println("stderr " + output);
}

@Override
public void stdoutWritten(String output) {
  // Pass error messages through flagging them as coming from stdout
  System.out.println("stdout " + output);
}
```

Save your changes and run your code.

If you try the Phobos plugin, browse to the location of the binary which will then be run and the output will appear in the console. If this is the first time you have run Phobos you will also need to look at the format since it will be necessary to parse this in order to extract the information needed for the annotations. The Phobos binary that you downloaded came with example repeat sequences which you can just drag into Geneious to test your plugin with.

Parsing the results

Hopefully your plugin correctly wrote out the FASTA sequence file and Phobos was able to read this and generate results which it will have written to `stdout` so it is now time to read those results and generate the annotations.

Looking only at `stdout`, you can see that the first line starts with a "#" symbol. This will be a good indicator that the output file has started. Also, each result starts with a line that has two spaces at the beginning which no other output line has so this is a good feature to look for.

Change the `stdoutWritten()` method to match the following:

```
public void stdoutWritten(String output) {
  if (output.startsWith("#")) {
    System.out.println("stdout  " + output);
  }
  else if (output.startsWith("  ")) {
    System.out.println("Found result: " + output);
  }
}
```

Save your changes, and run Phobos again. Try the Debug perspective and note that there is no need to relaunch Geneious as the code will hot swap. While you will still see all the messages that go to `stderr` and `stdout` but you should also see messages starting with '`Found result:`' from `stdout` which will be the result line starting with a space. You can put in break points and so on in the code to aid in debugging which you'll find useful as you work to develop future Geneious plugins.

Now that we have identified the exact line which has the information needed for the annotation it is necessary to use that to build an annotation.

As is the case with many command line applications, the output of Phobos appears at specific columns in the output and we can use this to extract the substrings to insert into the annotation. The most important features to look for are the start and stop positions for the annotation as well as the repeat class. These are the first three fields in the output line:

```
  hexanucleotide          3 :           49 |
```

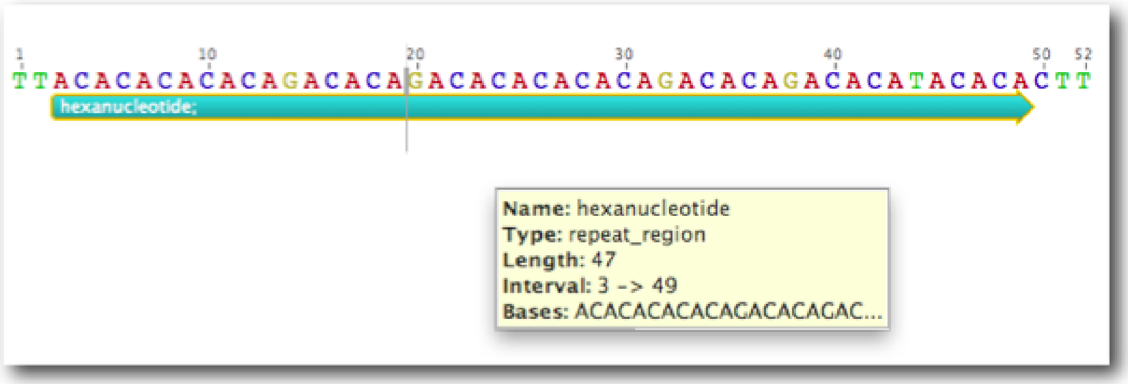
The first field finishes at position 16, the second at 27 and the third at 39. There are also some extra characters which we don't want so the actual range will be 0-16, 17-27 and 29-39.

Modify `stdoutWritten()` to match the following:

```
public void stdoutWritten(String output) {
  if (output.startsWith("#")) {
    System.out.println("stdout " + output);
  }
  else if (output.startsWith(" ")) {
    System.out.println("Found result: " + output);
    String repeatClass = output.substring(0, 16).trim();
    int startPos = Integer.parseInt(output.substring(17, 27).trim());
    int stopPos = Integer.parseInt(output.substring(29, 39).trim());

    final SequenceAnnotationInterval interval =
        new SequenceAnnotationInterval(startPos, stopPos);
    SequenceAnnotation annotation =
        new SequenceAnnotation(repeatClass,
                     SequenceAnnotation.TYPE_REPEAT_REGION, interval);
    results.addAnnotationToAdd(annotation);
  }
}
```

You can see that this code extracts the regions identified from the output and trims redundant spaces. It then uses the `startPos` and `stopPos` values to define the annotation interval and then creates an annotation.

If you run this code now, your plugin should be able to retrieve the location of the repeats and create annotations. Do that now and if there are repeats you should see something like this:



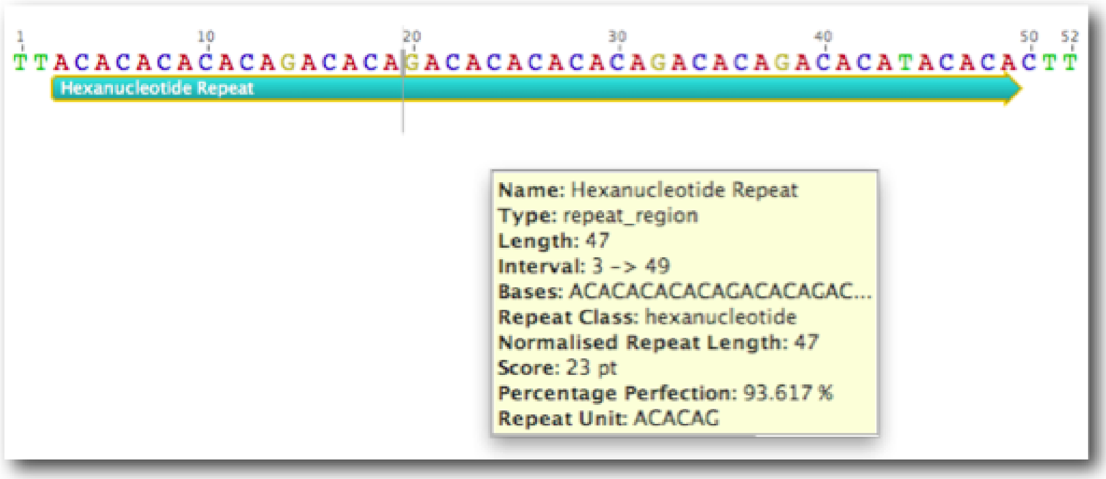
```
Name: hexanucleotide
Type: repeat_region
Length: 47
Interval: 3 -> 49
Bases: ACACACACACAGACACAGAC...
```

As you can see, the plugin has correctly extracted the repeat class, start and stop in this example.

There is much more useful information in the output so we can modify the parser to capture that as well as create a better name for the annotation:

```
else if (output.startsWith(" ")) {
  System.out.println("Found result: " + output);
  String repeatClass = output.substring(0, 16).trim();
  int startPos = Integer.parseInt(output.substring(17, 27).trim());
  int stopPos = Integer.parseInt(output.substring(29, 39).trim());
  String normalisedRepeatLength = output.substring(53, 60).trim();
  String score = output.substring(65, 75).trim();
  String percentage = output.substring(77, 87).trim();
  String repeatUnit = output.substring(94).trim();
  String repeatName = repeatClass.toUpperCase().substring(0, 1)
          + repeatClass.substring(1) + " Repeat";

  // Create new annotation and pass in the parsed data
  final SequenceAnnotationInterval interval =
      new SequenceAnnotationInterval(startPos, stopPos);
  SequenceAnnotation annotation =
      new SequenceAnnotation(repeatName,
                   SequenceAnnotation.TYPE_REPEAT_REGION, interval);
  annotation.addQualifier(new SequenceAnnotationQualifier("Repeat Class",
            repeatClass));
  annotation.addQualifier(new SequenceAnnotationQualifier("Normalised Repeat Length",
            normalisedRepeatLength));
  annotation.addQualifier(new SequenceAnnotationQualifier("Score", score));
  annotation.addQualifier(new SequenceAnnotationQualifier("Percentage Perfection",
            percentage));
  annotation.addQualifier(new SequenceAnnotationQualifier("Repeat Unit", repeatUnit));

  results.addAnnotationToAdd(annotation);
}
```

Note the change from `repeatClass` to `repeatName` in this version when the annotation is created and `repeatClass` is now added as a qualifier. Rerun Phobos to see the difference.



Name: Hexanucleotide Repeat
Type: repeat_region
Length: 47
Interval: 3 -> 49
Bases: ACACACACACAGACACAGAC...
Repeat Class: hexanucleotide
Normalised Repeat Length: 47
Score: 23 pt
Percentage Perfection: 93.617 %
Repeat Unit: ACACAG

Your plugin can now successfully run Phobos on one or more sequences and retrieve the results but has no progress indicator and it has no control over the optional parameters so the next step is to add a progress bar and then add more parameters to the options dialogue which currently only selects the location of the `phobos` binary.

Progress

Any plugin that will operate on a number of documents should show a progress bar. Now that your plugin runs this is the right time to add one.

Modify `PhobosAnnotationGenerator.java` to match the following:

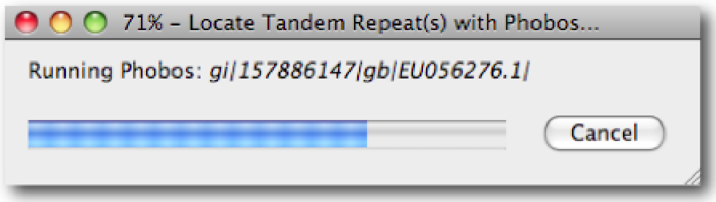
```
public List<SequenceAnnotationGenerator.AnnotationGeneratorResult>
generate(AnnotatedPluginDocument[] documents,
               SequenceAnnotationGenerator.SelectionRange selectionRange,
               ProgressListener progressListener, Options options)
               throws DocumentOperationException {

        List<SequenceAnnotationGenerator.AnnotationGeneratorResult> resultsList =
               new ArrayList<SequenceAnnotationGenerator.AnnotationGeneratorResult>();

        PhobosOptions phobosOptions = (PhobosOptions) options;

        // Create a progress bar calibrated to the number of documents
        CompositeProgressListener progress =
               new CompositeProgressListener(progressListener, documents.length);
```

Modify the `for` loop to match this next section:

```
for (AnnotatedPluginDocument annotatedPluginDocument:documents) {

  SequenceDocument seqDoc =
         (SequenceDocument) annotatedPluginDocument.getDocument();

  // Update progress bar with sequence name and progress
  progress.beginSubtask("Running Phobos: <i>" + seqDoc.getName() + "</i>");
```

Run this on a group of nucleotide documents.

Each time the loop is executed for a new document, the progress bar will update and inform the user of what document it is currently processing along with showing how far through the total job it has progressed:



Running Phobos: *gi|157886147|gb|EU056276.1|*

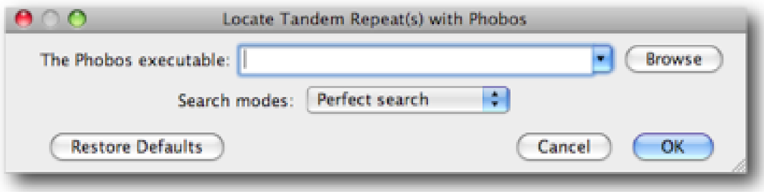
Add more options

Phobos provides a lot of options so we'll start by just adding a few. A simple one is the ability to specify exact or inexact matching. Phobos has three different possible states: *imperfect search*, *perfect search* and *extend exact search*. We'll now add a `ComboBoxOption` setting the default and providing the others for user selection. In addition, it is a good idea to add a description for what the option does which will appear as a tooltip for the user. Open the `PhobosOptions.java` file and modify it to match the following:

```
class PhobosOptions extends Options {

  public PhobosOptions() {
    codeLocationOptions();
    searchModeOptions();
  }

  private ComboBoxOption<OptionValue> searchModeOption;
  private static final String EXTEND_EXACT = "extendExact";
  private static final String EXACT = "exact";
  private static final String IMPERFECT = "imperfect";

  private void searchModeOptions() {
         Options.OptionValue[] searchModeComboBoxList = {
                new Options.OptionValue(IMPERFECT, "Imperfect search"), // Default
                new Options.OptionValue(EXACT, "Perfect search"),
                new Options.OptionValue(EXTEND_EXACT, "Extend exact search")};
         searchModeOption = addComboBoxOption("searchModes",
                "Search modes:", Arrays.asList(searchModeComboBoxList),
                searchModeComboBoxList[0]);
         searchModeOption.setDescription("Set search mode to imperfect (allowing
mismatches and gaps), perfect (exact repeats) or extend perfect repeats");
  }
```

It is good practice to use methods to separate the blocks of options into logical sets.

Since you've added a new method it is necessary to restart Geneious in Eclipse for your changes to show up - Eclipse will warn you that it can't do a hot code replacement so click the 'Restart' button in the dialogue. Once you have done that, if you run the plugin now you will get the following options dialogue:


Locate Tandem Repeat(s) with Phobos
The Phobos executable:
Browse
Search modes: Perfect search
Restore Defaults
Cancel
OK

You can select different search modes now but they will not be acted on until you add the necessary code to use the new option.

You will need to edit the `getCommand()` method and make the following change:

```
// Return the command line options
String[] getCommand(String tempFile) {
       ArrayList<String> commandList = new ArrayList<String>();

       commandList.add(codeLocation.getValue());
       commandList.add(tempFile);
       commandList.add("--searchMode " + searchModeOption.getValue().getName());

       // Make sure standard phobos output is used
       commandList.add("--outputFormat 0");
       // This will be the command used to execute
       return commandList.toArray(new String[commandList.size()]);
}
```

Note the additional `commandList.add()` which will insert your new command line options into the command line parameters passed back to `Execute`. Check the console to see that the search mode changes. Depending on the mode you should see one of these three:

```
#    Satellites search for:        imperfect repeats

#    Satellites search for:        exact repeats

#    Satellites search for:        exact repeats which are extended to imperfect repeats
```

Repeat masking

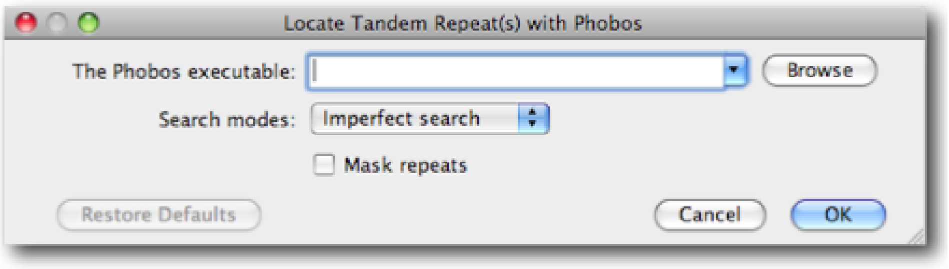
One of the reasons for detecting repeats is to be able to mask them. Currently, your plugin will annotated the repeat regions but to mask them you will need to change the sequence itself. Since the plugin has been developed around `SequenceAnnotationGenerator` you can also modify the bases that make up the sequence.

The changes to `PhobosOptions.java` are simple since this is a TRUE or FALSE operation (boolean). It needs a check box to toggle the feature so modify your `searchModeOptions()` method to match the following:

```
private ComboBoxOption<OptionValue> searchModeOption;
private static final String EXTEND_EXACT = "extendExact";
private static final String EXACT = "exact";
private static final String IMPERFECT = "imperfect";
private BooleanOption maskRepeatsOption;

private void searchModeOptions() {
       Options.OptionValue[] searchModeComboBoxList= {
              new Options.OptionValue(IMPERFECT, "Imperfect search"), // Default
              new Options.OptionValue(EXACT, "Perfect search"),
              new Options.OptionValue(EXTEND_EXACT, "Extend exact search")};
       searchModeOption = addComboBoxOption("searchModes",
              "Search modes:", Arrays.asList(searchModeComboBoxList),
              searchModeComboBoxList[0]);
       searchModeOption.setDescription("Set search mode to imperfect (allowing
mismatches and gaps), perfect (exact repeats) or extend perfect repeats");

       maskRepeatsOption = addBooleanOption("maskRepeats", "Mask repeats", false);
       maskRepeatsOption.setDescription("Mask repeats by replacing bases with 'N's");

}
```

This will add a new button that defaults to off.

Test it now and you should see this dialogue:



Unlike other options, we want Geneious to do the masking itself instead of having Phobos do it. This is because Geneious can modify the sequence and leave an editing record as an annotation. For this reason, there needs to be a method so other classes can get the value of the mask toggle. Add the `getMaskRepeatsOption()` method:

```
        maskRepeatsOption = addBooleanOption("maskRepeats", "Mask repeats", false);
        maskRepeatsOption.setDescription("Mask repeats by replacing bases with 'N's");
  }

  // Mask flag is needed by PhobosExecutionOutputListener
  boolean getMaskRepeatsOption() {
    return maskRepeatsOption.getValue();
  }
```

You now need to make some changes to `PhobosExecutionOutputListener.java` to access this mask flag and act on it.

```
class PhobosExecutionOutputListener extends Execution.OutputListener {

  private boolean maskFlag;

  public PhobosExecutionOutputListener(boolean maskFlag) {
    this.maskFlag = maskFlag;
  }
```

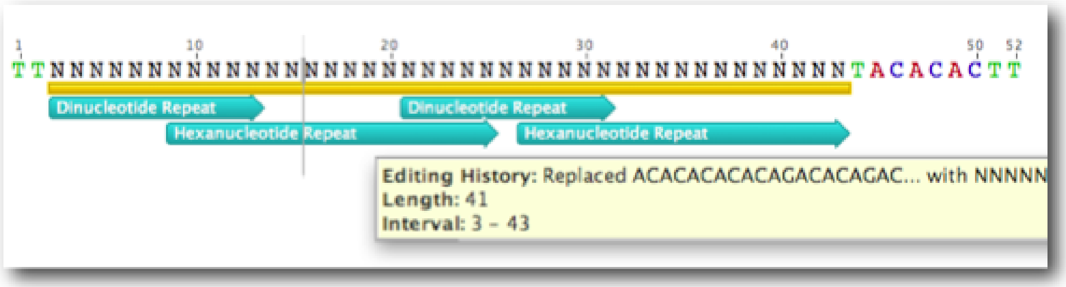
This creates a constructor allowing `maskFlag` to be passed in. The next step is to act on that flag using the start and stop co-ordinates for the repeat and to modify the bases in those positions. For the purposes of masking the bases can be changed to '`N`'. The following code should be added after the code that adds the annotation and it will create a string the length of the repeat made up of '`N`' characters which will then be inserted into the sequence. This operation edits the sequence and creates an edit annotation which contains the previous contents of the masked region.

```
annotation.addQualifier(new SequenceAnnotationQualifier("Percentage Perfection",
       percentage));
annotation.addQualifier(new SequenceAnnotationQualifier("Repeat Unit", RepeatUnit));

results.addAnnotationToAdd(annotation);

// Now mask out the bases if needed
if (maskFlag) {
  // Create the string of 'N's to insert
  StringBuilder maskedBases = new StringBuilder();
  for (int masking = 0; masking <= (stopPos - startPos); masking++) {
         maskedBases.append("N");
  }
  // Insert the N's into the sequence
  SequenceAnnotationGenerator.AnnotationGeneratorResult.ResidueAdjustment residueChanges =
         new SequenceAnnotationGenerator.AnnotationGeneratorResult.ResidueAdjustment(
                         startPos-1, stopPos, maskedBases.toString());
  results.addResidueAdjustment(residueChanges);
}
```

PhobosAnnotationGenerator.java will have an error which needs correcting. Locate the line indicated and change it to the following:

```
String[] command = phobosOptions.getCommand(tempFile.getAbsolutePath());

try {
  // Need a listener on the output stream
  PhobosExecutionOutputListener outputListener =
         new PhobosExecutionOutputListener(phobosOptions.getMaskRepeatsOption());
  // Build the Execution object with the command line, listeners
  // Nucleic Acid sequence
  Execution exec =
         new Execution(command, progressListener, outputListener, (String) null, false);
```

After making this change, test the code to see that it correctly masks the repeat regions. If it works you should see a result like this which was run using a perfect repeat search and with masking turned on:



Editing History: Replaced ACACACACACAGACACAGAC... with NNNNN
Length: 41
Interval: 3 - 43

As you can see, mousing over the masked region brings up a tool tip showing the previous bases before masking.

Remove hidden repeats

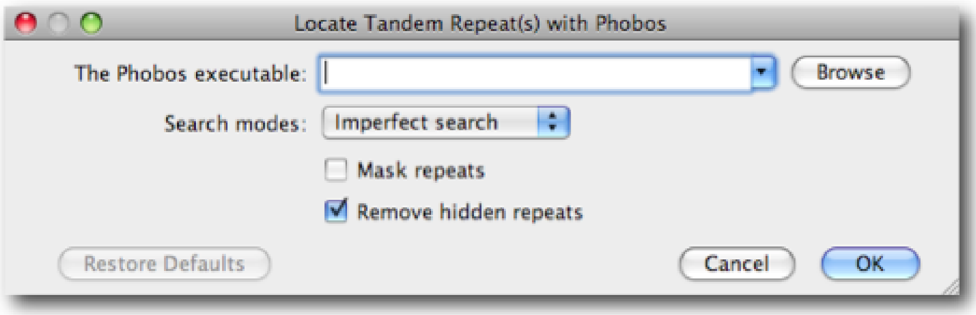
Another simple boolean option which can be added. This needs no more work than to add the option and make sure the command line is used. In PhobosOptions.java modify your previously defined searchModeOptions() method as follows:

```
private ComboBoxOption<OptionValue> searchModeOption;
private static final String EXTEND_EXACT = "extendExact";
private static final String EXACT = "exact";
private static final String IMPERFECT = "imperfect";
private BooleanOption maskRepeatsOption;
private BooleanOption removeHiddenOption;

private void searchModeOptions() {
       Options.OptionValue[] searchModeComboBoxList= {
              new Options.OptionValue(IMPERFECT, "Imperfect search"), // Default
              new Options.OptionValue(EXACT, "Perfect search"),
              new Options.OptionValue(EXTEND_EXACT, "Extend exact search")};
       searchModeOption = addComboBoxOption("searchModes",
              "Search modes:", Arrays.asList(searchModeComboBoxList),
              searchModeComboBoxList[0]);
       searchModeOption.setDescription("Set search mode to imperfect (allowing
mismatches and gaps), perfect (exact repeats) or extend perfect repeats");

       maskRepeatsOption = addBooleanOption("maskRepeats", "Mask repeats", false);
       maskRepeatsOption.setDescription("Mask repeats by replacing bases with 'N's");

       removeHiddenOption = addBooleanOption("removeHidden", "Remove hidden repeats", true);
       removeHiddenOption.setDescription("Remove repeats that are mostly overlapped by or
do mostly overlap an higher scoring repeat");

}
```

Now you just need to add code to handle the option so add the indicated lines to your `getCommand()` method.

```
// Return the command line options
String[] getCommand(String tempFile) {
       ArrayList<String> commandList = new ArrayList<String>();

       commandList.add(codeLocation.getValue());
       commandList.add(tempFile);
       commandList.add("--searchMode " + searchModeOption.getValue().getName());
       if (!removeHiddenOption.getValue()) {
         commandList.add("--dontRemoveMostlyOverlapping");
       }

       // Make sure standard phobos output is used
       commandList.add("--outputFormat 0");
       // This will be the command used to execute
       return commandList.toArray(new String[commandList.size()]);
}
```

Run the code and check the console to see that the option is correctly being applied.



The Phobos executable:

Search modes: Imperfect search

☐ Mask repeats

☑ Remove hidden repeats

So far, with the few options you have added, the dialogue has come out looking pretty good. Each option is self contained and the layout looks good too. Unfortunately, as you add more options it is likely that things will get messy and handling options which depend on others as well as making the layout work adds new complications.

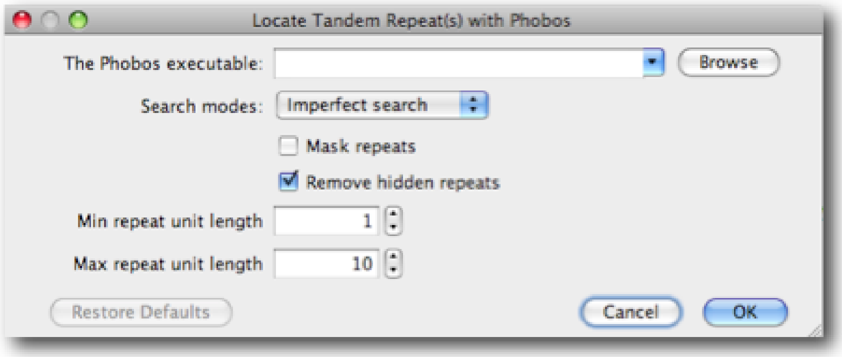
Min and Max repeat unit lengths

Phobos allows the user to specify a range of repeat unit lengths so your GUI should do the same. Make the following changes to your `searchModeOptions()` method:

```
private BooleanOption maskRepeatsOption;
private BooleanOption removeHiddenOption;
private IntegerOption minUnitLenOption;
private IntegerOption maxUnitLenOption;

private void searchModeOptions() {
     Options.OptionValue[] searchModeComboBoxList= {
            new Options.OptionValue(IMPERFECT, "Imperfect search"), // Default
            new Options.OptionValue(EXACT, "Perfect search"),
            new Options.OptionValue(EXTEND_EXACT, "Extend exact search")};
     searchModeOption = addComboBoxOption("searchModes",
            "Search modes:", Arrays.asList(searchModeComboBoxList),
            searchModeComboBoxList[0]);
     searchModeOption.setDescription("Set search mode to imperfect (allowing
mismatches and gaps), perfect (exact repeats) or extend perfect repeats");

     maskRepeatsOption = addBooleanOption("maskRepeats", "Mask repeats", false);
     maskRepeatsOption.setDescription("Mask repeats by replacing bases with 'N's");

     removeHiddenOption = addBooleanOption("removeHidden", "Remove hidden repeats", true);
     removeHiddenOption.setDescription("Remove repeats that are mostly overlapped by or
do mostly overlap an higher scoring repeat");

     minUnitLenOption = addIntegerOption("minUnitLen", "Min repeat unit length", 1, 1, 1000);
     maxUnitLenOption = addIntegerOption("maxUnitLen", "Max repeat unit length", 10, 1, 1000);
     minUnitLenOption.setDescription("Minimum unit (pattern) size in search.  Default: 1");
```

```
    maxUnitLenOption.setDescription("Maximum unit (pattern) size in search.  Default: 10");
}
```

Before adding any code to act on these new options, run to see how the options dialogue is looking.

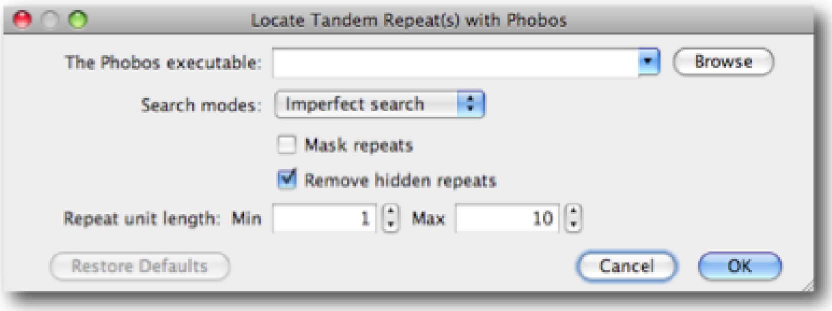


The Phobos executable:
Search modes: Imperfect search
[ ] Mask repeats
[✓] Remove hidden repeats
Min repeat unit length 1
Max repeat unit length 10

While this doesn't look too bad the labels are rather repetitive so it would make sense to try and use just one label. Also, there is a lot of dead space on the right of these new options which would be nice to use. Fortunately, you can have multiple options on a single line so modify the code you just added to match the following:

```
beginAlignHorizontally("Repeat unit length: Min", false);
minUnitLenOption = addIntegerOption("minUnitLen", "", 1, 1, 1000);
maxUnitLenOption = addIntegerOption("maxUnitLen", " Max", 10, 1, 1000);
minUnitLenOption.setDescription("Minimum unit (pattern) size in search.  Default: 1");
maxUnitLenOption.setDescription("Maximum unit (pattern) size in search.  Default: 10");
endAlignHorizontally();
```

Note the addition of `beginAlignHorizontally` and ěndAlignHorizontally. Also, note that you can include a label in the `beginAlignHorizontally` which is useful in this case as it allows us to use the available space better for example, if the '`Min`' was used in the label for the `minUnitLenOption` it would pad the input box over and it is nice to keep it in line with the other options (but try it anyway if you like).



The Phobos executable:

Search modes: Imperfect search

[ ] Mask repeats

[✓] Remove hidden repeats

Repeat unit length: Min 1 Max 10

This layout is more pleasing since there is less wasted space in the dialogue and as we wanted, the `min` and `max` options are now side by side.

As before, you also need to add these two options to the `getCommand()` method.

```
commandList.add("--searchMode " + searchModeOption.getValue().getName());
if (!removeHiddenOption.getValue()) {
  commandList.add("--dontRemoveMostlyOverlapping");
}
commandList.add("--minUnitLen " + minUnitLenOption.getValue());
commandList.add("--maxUnitLen " + maxUnitLenOption.getValue());
```

Now, if you test the code you should be able to change these values and Phobos will react accordingly. Check the console output to see that it recognised your values for `min` and `max`.

However, there is still the issue of the `min` and `max` values being linked and your GUI allows you to set `min` to a value greater than `max` and `max` to a value less than `min`. This does not make sense so you need to add code to handle this scenario. The way to do this is to use a `SimpleListener`. Add the following code after your options are declared in `PhobosOptions()`.

```
public PhobosOptions() {
        codeLocationOptions();
        searchModeOptions();

        // Change Listeners

        minUnitLenOption.addChangeListener(new SimpleListener() {
                public void objectChanged() {
                        if (minUnitLenOption.getValue() > maxUnitLenOption.getValue()) {
                                maxUnitLenOption.setValue(minUnitLenOption.getValue());
                        }
                 }
         });

         maxUnitLenOption.addChangeListener(new SimpleListener() {
              public void objectChanged() {
                     if (maxUnitLenOption.getValue() < minUnitLenOption.getValue()) {
                            minUnitLenOption.setValue(maxUnitLenOption.getValue());
                     }
              }
       });
}
```

What this code is doing is listening for an event on one of the options and then does a check to see that the change has gone beyond the partner option. If it has, then it changes the partner to be the same value. If you test this code you should now find that as you dial the `min` value up past the `max` value then `max` starts to move up with it. Similarly, if you drop the `max` value below the `min`, then `min` will go down at the same time. This way the `min` value can only every be less than or equal to `max` and similarly, `max` can only every be more than or equal to `min`.

Imperfect search options

Phobos allows the user to set various options for the imperfect search so we should implement these now. As with the previous `min` and `max` values, some of these will be linked but also we need to implement some presets. First add a call to a new method called `imperfectSearchOptions()` in `PhobosOptions()`.

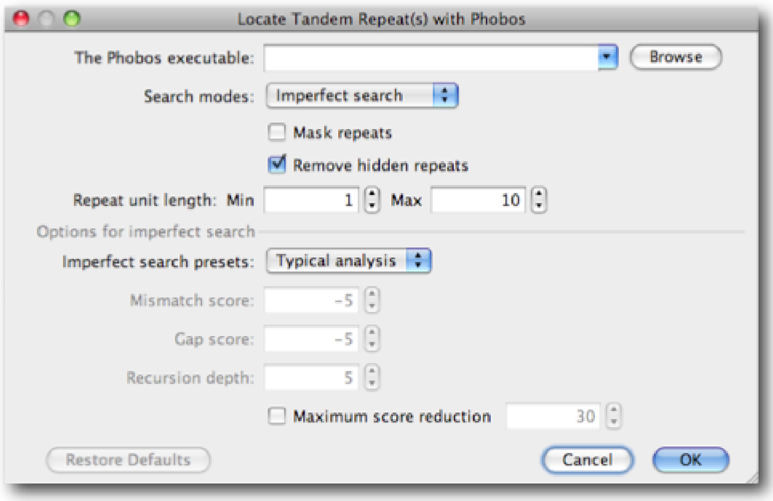
```
public PhobosOptions() {
       codeLocationOptions();
       searchModeOptions();
       imperfectSearchOptions();

       // Change listeners
```

Next, add the code for the method itself to the `PhobosOptions` class:

```
private static final String MASKING = "masking";
private static final String ANALYSIS = "analysis";
private static final String CUSTOM = "custom";
private ComboBoxOption<OptionValue> typicalOption;
private IntegerOption mismatchScoreOption;
private IntegerOption gapScoreOption;
private IntegerOption recursionDepthOption;
private BooleanOption scoreReductionOption;
private IntegerOption scoreReductionValueOption;
private void imperfectSearchOptions() {
  addDivider("Options for imperfect search");

  Options.OptionValue[] typicalOptionsComboBoxList = {
```

```
                  new Options.OptionValue(CUSTOM, "Custom settings"),
                  new Options.OptionValue(ANALYSIS, "Typical analysis"),
                  new Options.OptionValue(MASKING, "Typical masking")
  };

  typicalOption = addComboBoxOption("typicalOptions", "Imperfect search presets:",
                  Arrays.asList(typicalOptionsComboBoxList), typicalOptionsComboBoxList[1]);

  mismatchScoreOption = addIntegerOption("mismatchScore", "Mismatch score:", -5, -999, -2);
  mismatchScoreOption.setDescription("Usually -3 to -6");
  mismatchScoreOption.setEnabled(false);
  gapScoreOption = addIntegerOption("gapScore", "Gap score:", -5, -999, -2);
  gapScoreOption.setDescription("Usually -3 to -6");
  gapScoreOption.setEnabled(false);
  recursionDepthOption = addIntegerOption("recursionDepth", "Recursion depth:", 5, 0, 1000);
  recursionDepthOption.setEnabled(false);

  beginAlignHorizontally("", false);
  scoreReductionOption = addBooleanOption("scoreReduction", "Maximum score reduction", false);
  scoreReductionValueOption = addIntegerOption("scoreReductionValue",
                  "", 30, 0, 100000);
  scoreReductionOption.setDescription("Maximum score reduction allowed in search e.g. 30.
Default: off");
  scoreReductionValueOption.setDescription("Maximum score reduction allowed in search e.g.
30.  Default: off");
  scoreReductionOption.addDependent(scoreReductionValueOption, true);
  endAlignHorizontally();
}
```

There is quite a lot of new code here and we're not finished yet but it should look quite similar to code you've already written since there is another `comboBox` and some more `Boolean` and `Integer` options. You have also seen the horizontal alignment but we've also now added a divider to identify this group of options. Run the code and you should now have a dialogue with imperfect search options.


Locate Tandem Repeat(s) with Phobos

The Phobos executable:

Search modes: Imperfect search

☐ Mask repeats

☑ Remove hidden repeats

Repeat unit length: Min 1 Max 10

Options for imperfect search

Imperfect search presets: Typical analysis

Mismatch score: -5

Gap score: -5

Recursion depth: 5

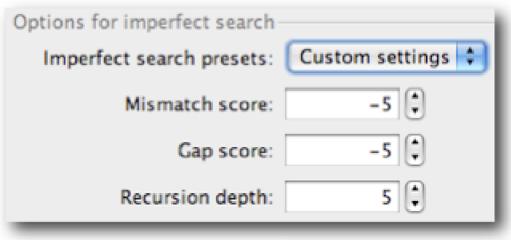
☐ Maximum score reduction 30

Notice how some of these options are disabled and greyed out. As we add more code to deal with the presets we will selectively enable these options. Also, we will need to be able to disable all of these options in the case of a perfect search. Add the following code after your previous listeners.

```
maxUnitLenOption.addChangeListener(new SimpleListener() {
        public void objectChanged() {
                if (maxUnitLenOption.getValue() < minUnitLenOption.getValue()) {
                        minUnitLenOption.setValue(maxUnitLenOption.getValue());
                }
        }
});

typicalOption.addChangeListener(new SimpleListener() {
        public void objectChanged() {
                if (typicalOption.getValue().getName().equals(CUSTOM)) {
                        mismatchScoreOption.setEnabled(true);
                        gapScoreOption.setEnabled(true);
                        recursionDepthOption.setEnabled(true);
                }
                else if (typicalOption.getValue().getName().equals(ANALYSIS)) {
                        mismatchScoreOption.setValue(-5);
                        mismatchScoreOption.setEnabled(false);
                        gapScoreOption.setValue(-5);
                        gapScoreOption.setEnabled(false);
                        recursionDepthOption.setValue(5);
                        recursionDepthOption.setEnabled(false);
                }
                else if (typicalOption.getValue().getName().equals(MASKING)) {
                        mismatchScoreOption.setValue(-4);
                        mismatchScoreOption.setEnabled(false);
                        gapScoreOption.setValue(-4);
                        gapScoreOption.setEnabled(false);
                        recursionDepthOption.setValue(3);
                        recursionDepthOption.setEnabled(false);
                }
        }
});
```

This code will enable the `IntegerOption` boxes when Custom is selected, and disable them while setting them to appropriate values for Typical Analysis and Typical Masking.

Test that it behaves correctly.



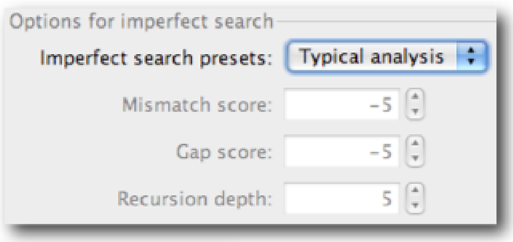
Options for imperfect search

Imperfect search presets: Custom settings

Mismatch score: -5

Gap score: -5

Recursion depth: 5



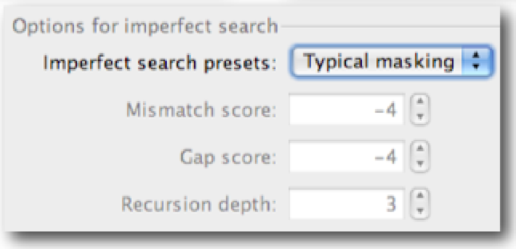
Options for imperfect search

Imperfect search presets: Typical analysis

Mismatch score: -5

Gap score: -5

Recursion depth: 5



Options for imperfect search

Imperfect search presets: Typical masking

Mismatch score: -4

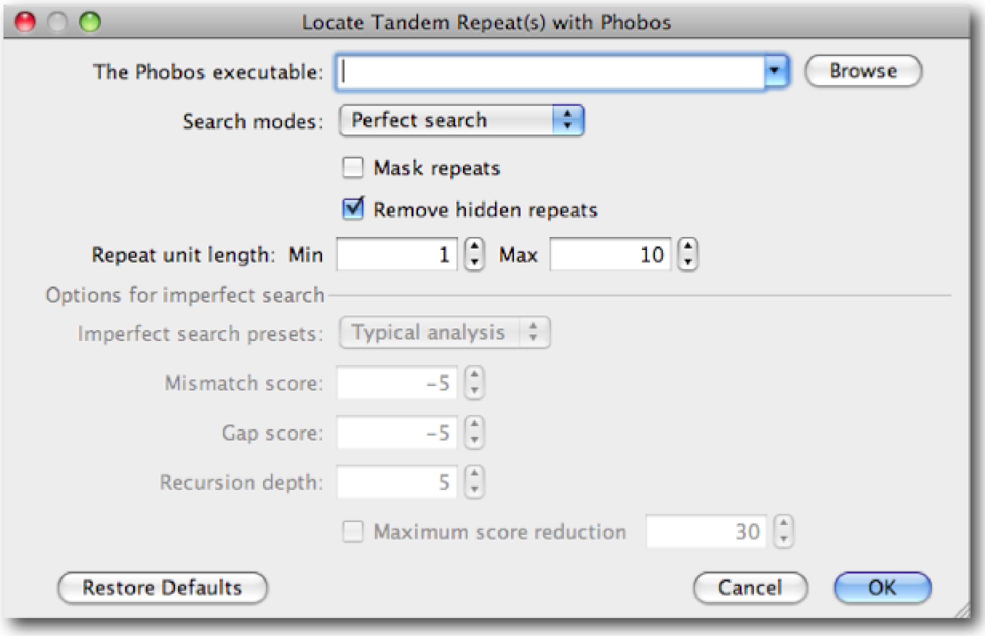
Gap score: -4

Recursion depth: 3

Next we need to make sure that all of these options are disabled in the case of a '`Perfect Search`'. Add the following code just after the previous section's code:

```
                        gapScoreOption.setValue(-4);
                        gapScoreOption.setEnabled(false);
                        recursionDepthOption.setValue(3);
                        recursionDepthOption.setEnabled(false);
                }
        }
});

searchModeOption.addChangeListener(new SimpleListener() {
        public void objectChanged() {
                if (searchModeOption.getValue().getName().equals(EXACT)) {
                        // Disable gap options for perfect mode
                        typicalOption.setEnabled(false);
                        mismatchScoreOption.setEnabled(false);
                        gapScoreOption.setEnabled(false);
                        recursionDepthOption.setEnabled(false);
                        scoreReductionOption.setEnabled(false);
                }
                else if (searchModeOption.getValue().getName().equals(IMPERFECT)
                        || searchModeOption.getValue().getName().equals(EXTEND_EXACT)) {
                        typicalOption.setEnabled(true);
                        mismatchScoreOption.setEnabled(true);
                        gapScoreOption.setEnabled(true);
                        recursionDepthOption.setEnabled(true);
                        scoreReductionOption.setEnabled(true);
                }
        }
});
```

You can now test that the options are disabled and enabled as expected when '`Perfect Search`' is selected and if they are you should see the following:


Locate Tandem Repeat(s) with Phobos

The Phobos executable:

Search modes: Perfect search

[ ] Mask repeats

[✓] Remove hidden repeats

Repeat unit length: Min 1 Max 10

Options for imperfect search

Imperfect search presets: Typical analysis

Mismatch score: -5

Gap score: -5

Recursion depth: 5

[ ] Maximum score reduction 30

The final step is to make these new options active by adding the following sections of code to the `getCommand()` method.

```
commandList.add("--minUnitLen " + minUnitLenOption.getValue());
commandList.add("--maxUnitLen " + maxUnitLenOption.getValue());

if (scoreReductionOption.getValue()) {
       commandList.add("--maximum_score_reduction " + scoreReductionValueOption.getValue());
}
commandList.add("--mismatchScore " + mismatchScoreOption.getValue());
commandList.add("--indelScore " + gapScoreOption.getValue());
commandList.add("--recursion " + recursionDepthOption.getValue());

// Make sure standard Phobos output is used
commandList.add("--outputFormat 0");
```

Run this and check that the new options are changing the Phobos settings correctly by examining the output on `stdout` in the console.

Satellite Options and ‘N’ Handling

We’re in the home straight as far as the options go so now we’re going to push on to the end and add all remaining options. Add the following method calls to `PhobosOptions()`:

```
public PhobosOptions() {
        codeLocationOptions();
        searchModeOptions();
        imperfectSearchOptions();
        satelliteRequirementsOptions();
        nHandlingOptions();

        // Change listeners

        minUnitLenOption.addChangeListener(new SimpleListener() {
```

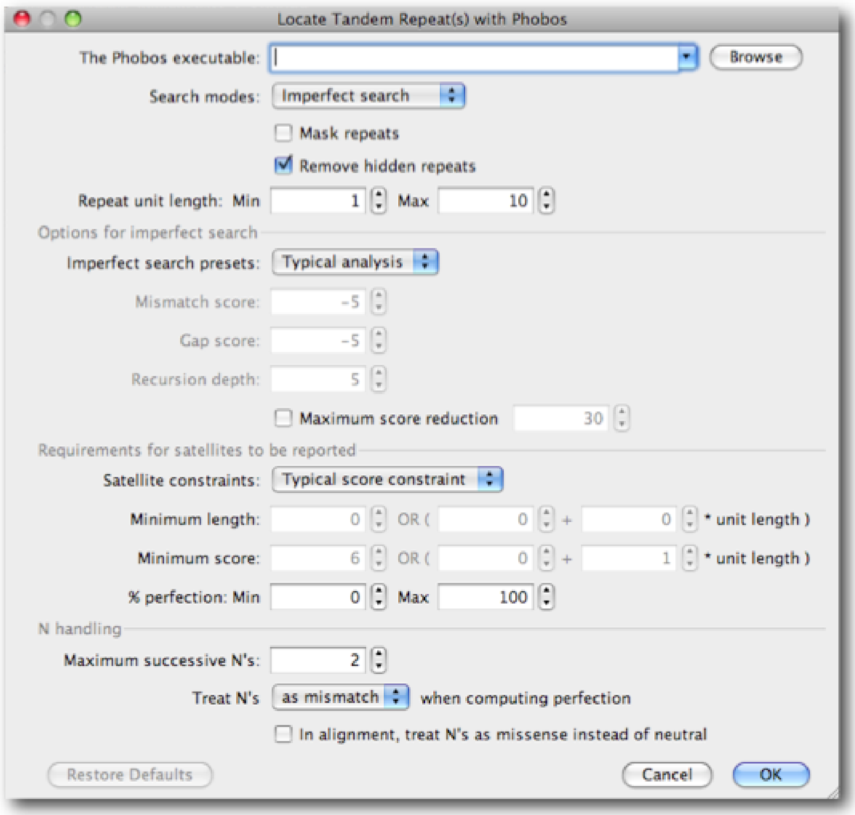
Add this to the `PhobosOptions` class defining `satelliteRequirementsOptions()`:

```
private static final String LENGTH_CONSTRAINT = "lengthConstraint";
private static final String SCORE_CONSTRAINT = "scoreConstraint";
private static final String CUSTOM_CONSTRAINT = "customConstraint";
private ComboBoxOption<OptionValue> typicalConstraintsOption;
private IntegerOption minScoreSatOption;
private IntegerOption minScoreSatAOption;
private DoubleOption minScoreSatBOption;
private IntegerOption minLengthSatOption;
private IntegerOption minLengthSatAOption;
private DoubleOption minLengthSatBOption;
private IntegerOption minPerfectionOption;
private IntegerOption maxPerfectionOption;

private void satelliteRequirementsOptions() {
  addDivider("Requirements for satellites to be reported");

  final Options.OptionValue[] typicalConstraintsComboBoxList = {
         new Options.OptionValue(CUSTOM_CONSTRAINT, "Custom constraints"),
         new Options.OptionValue(SCORE_CONSTRAINT, "Typical score constraint"), // Default
         new Options.OptionValue(LENGTH_CONSTRAINT, "Typical length constraint")};
  typicalConstraintsOption = addComboBoxOption("constraintsOptions", "Satellite constraints:",
         Arrays.asList(typicalConstraintsComboBoxList),typicalConstraintsComboBoxList[1]);

  beginAlignHorizontally("Minimum length:", false);
  minLengthSatOption = addIntegerOption("minLengthSat", "", 0, 0, 1000000);
  minLengthSatOption.setEnabled(false);
  minLengthSatAOption = addIntegerOption("minLengthSatA", " OR (", 0, 0, 1000000);
  minLengthSatAOption.setEnabled(false);
  minLengthSatBOption = addDoubleOption("minLengthSatB", "+", 0.0, 0.0, 1000000.0);
  minLengthSatBOption.setEnabled(false);
  addLabel("* unit length )");
  minLengthSatOption.setDescription("Sets the minimum length of a satellite using the first
value, or the result of the second calculation, whichever is larger");
  minLengthSatAOption.setDescription("Sets the minimum length of a satellite using the first
value, or the result of the second calculation, whichever is larger");
  minLengthSatBOption.setDescription("Sets the minimum length of a satellite using the first
value, or the result of the second calculation, whichever is larger");
  endAlignHorizontally();
  beginAlignHorizontally("Minimum score:", false);
  minScoreSatOption = addIntegerOption("minScoreSat", "", 6, 0, 1000000);
  minScoreSatOption.setEnabled(false);
  minScoreSatAOption = addIntegerOption("minScoreSatA", " OR (", 0, 0, 1000000);
  minScoreSatAOption.setEnabled(false);
  minScoreSatBOption = addDoubleOption("minScoreSatB", "+", 1.0, 0.0, 1000000.0);
  minScoreSatBOption.setEnabled(false);
  addLabel("* unit length )");
  minScoreSatOption.setDescription("Sets the minimum score of a satellite using the first
value, or the result of the second calculation, whichever is larger");
  minScoreSatAOption.setDescription("Sets the minimum score of a satellite using the first
value, or the result of the second calculation, whichever is larger");
  minScoreSatBOption.setDescription("Sets the minimum score of a satellite using the first
value, or the result of the second calculation, whichever is larger");
  endAlignHorizontally();

  beginAlignHorizontally("% perfection: Min", false);
  minPerfectionOption = addIntegerOption("minPerfection", "", 0, 0, 100);
  maxPerfectionOption = addIntegerOption("maxPerfection", " Max", 100, 0, 100);
  minPerfectionOption.setDescription("Minimum perfection of a repeat. Default: 0%");
  maxPerfectionOption.setDescription("Maximum perfection of a repeat. Default: 100%");
  endAlignHorizontally();
}
```

Similarly, add the following to define `nHandlingOptions()`:

```
private IntegerOption maxNOption;
private ComboBoxOption<OptionValue> nModeOption;
private BooleanOption missenseOption;

private void nHandlingOptions() {
       addDivider("N handling");
       maxNOption = addIntegerOption("maxN", "Maximum successive N's:", 2, 0, 1000000);
       maxNOption.setDescription("Maximum number of successive N's allowed in a repeat");
       beginAlignHorizontally("Treat N's", false);
       Options.OptionValue[] nModeComboBoxList = {
              new Options.OptionValue("0", "as mismatch"), // Default
              new Options.OptionValue("1", "as neutral"),
              new Options.OptionValue("2", "as match") };
       nModeOption = addComboBoxOption("nMode", "",
              Arrays.asList(nModeComboBoxList), nModeComboBoxList[0]);
       addLabel("when computing perfection");
       endAlignHorizontally();
       missenseOption = addBooleanOption("treatNs", "In alignment, treat N's as
missense instead of neutral", false);
       missenseOption.addDependent(nModeOption, false);
}
```

Run this and you will get a greatly expanded options dialogue which looks like this:


Locate Tandem Repeat(s) with Phobos

The Phobos executable: | Browse

Search modes: Imperfect search

☐ Mask repeats

☑ Remove hidden repeats

Repeat unit length: Min 1 Max 10

Options for imperfect search

Imperfect search presets: Typical analysis

Mismatch score: -5

Gap score: -5

Recursion depth: 5

☐ Maximum score reduction 30

Requirements for satellites to be reported

Satellite constraints: Typical score constraint

Minimum length: 0 OR (0 + 0 * unit length)

Minimum score: 6 OR (0 + 1 * unit length)

% perfection: Min 0 Max 100

N handling

Maximum successive N's: 2

Treat N's as mismatch when computing perfection

☐ In alignment, treat N's as missense instead of neutral

Restore Defaults Cancel OK

Again, there need to be some listeners added to handle the '`% perfection`'options since we don't want `Min` to be higher than `Max` or vice versa. Put these with your other listeners in `PhobosOptions()`.

```
minPerfectionOption.addChangeListener(new SimpleListener() {
        public void objectChanged() {
                if (minPerfectionOption.getValue() > maxPerfectionOption.getValue()) {
                        maxPerfectionOption.setValue(minPerfectionOption.getValue());
                }
        }
});

maxPerfectionOption.addChangeListener(new SimpleListener() {
        public void objectChanged() {
                if (maxPerfectionOption.getValue() < minPerfectionOption.getValue()) {
                        minPerfectionOption.setValue(maxPerfectionOption.getValue());
                }
        }
});
```

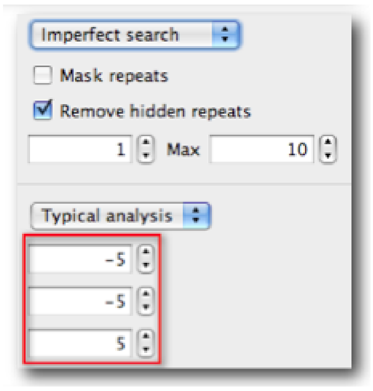
It will also be necessary to add a listener for the '`Satellite constraints`' options which again will behave quite like that previously added for the '`Imperfect search presets`'.

```
typicalConstraintsOption.addChangeListener(new SimpleListener() {
   public void objectChanged() {
       if (typicalConstraintsOption.getValue().getName().equals(CUSTOM_CONSTRAINT)) {
             minScoreSatOption.setEnabled(true);
             minScoreSatAOption.setEnabled(true);
             minScoreSatBOption.setEnabled(true);
             minLengthSatOption.setEnabled(true);
             minLengthSatAOption.setEnabled(true);
             minLengthSatBOption.setEnabled(true);
       }
       else if (typicalConstraintsOption.getValue().getName().equals(LENGTH_CONSTRAINT)) {
             minScoreSatOption.setValue(0);
             minScoreSatOption.setEnabled(false);
             minScoreSatAOption.setValue(0);
             minScoreSatAOption.setEnabled(false);
             minScoreSatBOption.setValue(0.0);
             minScoreSatBOption.setEnabled(false);
             minLengthSatOption.setValue(8);
             minLengthSatOption.setEnabled(false);
             minLengthSatAOption.setValue(0);
             minLengthSatAOption.setEnabled(false);
             minLengthSatBOption.setValue(2.0);
             minLengthSatBOption.setEnabled(false);
       }
       else if (typicalConstraintsOption.getValue().getName().equals(SCORE_CONSTRAINT)) {
             minScoreSatOption.setValue(6);
             minScoreSatOption.setEnabled(false);
             minScoreSatAOption.setValue(0);
             minScoreSatAOption.setEnabled(false);
             minScoreSatBOption.setValue(1.0);
             minScoreSatBOption.setEnabled(false);
             minLengthSatOption.setValue(0);
             minLengthSatOption.setEnabled(false);
             minLengthSatAOption.setValue(0);
             minLengthSatAOption.setEnabled(false);
             minLengthSatBOption.setValue(0.0);
             minLengthSatBOption.setEnabled(false);
         }
    }
});
```

Finally, change the previously added `searchModeOption` listener to handle the perfection settings:

```
searchModeOption.addChangeListener(new SimpleListener() {
       public void objectChanged() {
              if (searchModeOption.getValue().getName().equals(EXACT)) {
                     // Disable gap options for perfect mode
                     minPerfectionOption.setEnabled(false);
                     maxPerfectionOption.setEnabled(false);
                     typicalOption.setEnabled(false);
                     mismatchScoreOption.setEnabled(false);
                     gapScoreOption.setEnabled(false);
                     recursionDepthOption.setEnabled(false);
                     scoreReductionOption.setEnabled(false);
              }
              else if (searchModeOption.getValue().getName().equals(IMPEFECT))
                     || searchModeOption.getValue().getName().equals(EXTEND_EXACT)) {
                     minPerfectionOption.setEnabled(true);
                     maxPerfectionOption.setEnabled(true);
                     typicalOption.setEnabled(true);
                     mismatchScoreOption.setEnabled(true);
                     gapScoreOption.setEnabled(true);
                     recursionDepthOption.setEnabled(true);
                     scoreReductionOption.setEnabled(true);
              }
       }
});
```

Interestingly, there is a bug in this code. Enabling the mismatch, gap, recursion and score reduction options for imperfect modes is only correct if the options are set to custom and not typical masking and analysis. Try changing between perfect and imperfect modes with and without custom options set and you will see it is possible to get into a state where the integer option boxes are active even in masking or analysis mode.



To correct this, change the above code to to the following:

```
else if (searchModeOption.getValue().getName().equals(IMPERFECT)
        || searchModeOption.getValue().getName().equals(EXTEND_EXACT)) {
                minPerfectionOption.setEnabled(true);
                maxPerfectionOption.setEnabled(true);
                typicalOption.setEnabled(true);
                // Make sure these options are only enabled if CUSTOM was set
                if (typicalOption.getValue().getName().equals(CUSTOM)) {
                        mismatchScoreOption.setEnabled(true);
                        gapScoreOption.setEnabled(true);
                        recursionDepthOption.setEnabled(true);
                }
                else {
                        mismatchScoreOption.setEnabled(false);
                        gapScoreOption.setEnabled(false);
                        recursionDepthOption.setEnabled(false);
                }
                scoreReductionOption.setEnabled(true);
        }
```

You should now see the behaviour is correct. When enabling and disabling options you need to take great care that even subtle bugs like this are dealt with.

Now update your `getCommand()` method to handle the new options:

```
// Return the command line options
String[] getCommand(String tempFile) {
        ArrayList<String> commandList = new ArrayList<String>();
        commandList.add(codeLocation.getValue());
        commandList.add(tempFile);
        commandList.add("--searchMode " + searchModeOption.getValue().getName());
        if (!removeHiddenOption.getValue()) {
                commandList.add("--dontRemoveMostlyOverlapping");
        }
        commandList.add("--minUnitLen " + minUnitLenOption.getValue());
        commandList.add("--maxUnitLen " + maxUnitLenOption.getValue());

        if (scoreReductionOption.getValue()) {
                commandList.add("--maximum_score_reduction " +
                        scoreReductionValueOption.getValue());
        }
        commandList.add("--mismatchScore " + mismatchScoreOption.getValue());
        commandList.add("--indelScore " + gapScoreOption.getValue());
        commandList.add("--recursion " + recursionDepthOption.getValue());

        commandList.add("--NPerfectionMode " + nModeOption.getValue().getName());
        if (missenseOption.getValue()) {
                commandList.add("--NsAsMissense");
        }
        commandList.add("--succN " + maxNOption.getValue());
        commandList.add("--minScore " + minScoreSatOption.getValue());
        commandList.add("--minScore_a " + minScoreSatAOption.getValue());
        commandList.add("--minScore_b " + minScoreSatBOption.getValue());
        commandList.add("--minLength " + minLengthSatOption.getValue());
        commandList.add("--minLength_a " + minLengthSatAOption.getValue());
        commandList.add("--minLength_b " + minLengthSatBOption.getValue());
        // These two should be floats but to make the GUI nicer they are really
        // integers so put a .0 on the end
        commandList.add("--maxPerfection " + maxPerfectionOption.getValue() + ".0");
        commandList.add("--minPerfection " + minPerfectionOption.getValue() + ".0");

        // Make sure standard Phobos output is used
        commandList.add("--outputFormat 0");
        // This will be the command used by execute
        return commandList.toArray(new String[commandList.size()]);
}
```

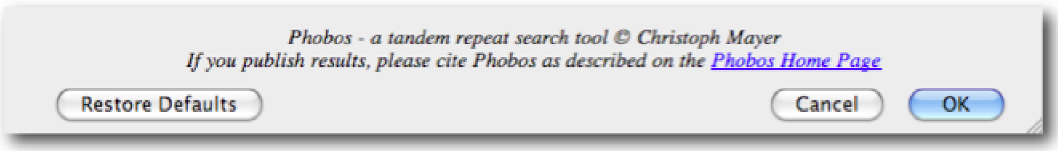
As usual, check that Phobos is getting the correct options passed to it.

Some Nice Additions

At this point, you have a fully functional implementation of the the Phobos plugin but there are some final improvements to make which can be useful. Since this plugin is a wrapper for a command line program, it would be nice to give the original author credit for their work. A good way to do this is to add the following code to `PhobosOptions.java` and add the appropriate imports.

```
private final String PHOBOS_URL = "http://www.rub.de/spezzoo/cm/cm_phobos.htm";
// Create a custom citation
protected JPanel createPanel() {
       JPanel mainPanel = new JPanel(new BorderLayout());
       JPanel defaultPanel = super.createPanel();
       mainPanel.add(defaultPanel, BorderLayout.CENTER);
       GEditorPane citationPane = new GEditorPane();
       citationPane.setEditable(false);
       citationPane.setOpaque(false);
       citationPane.setContentType("text/html");
       citationPane.addHyperlinkListener(new SimpleLinkListener());
       citationPane.setText("<html><br><i><center>Phobos - a tandem repeat search tool &copy;
Christoph Mayer<br>"
+ "If you publish results, please cite Phobos as described on the <a href=\""
+ PHOBOS_URL + "\">Phobos Home Page</a></center></i></html>");
       mainPanel.add(citationPane, BorderLayout.SOUTH);

       return mainPanel;
}
```

Compile and you should see the following added to the bottom of your options dialogue.


Phobos - a tandem repeat search tool © Christoph Mayer
If you publish results, please cite Phobos as described on the Phobos Home Page

Another useful feature is to add a check that the correct version of Phobos is installed and also add information on how to get it via a 'Help' button. First, make sure you have all of the following imports at the top of `PhobosOptions.java`.

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.util.ArrayList;
import java.util.Arrays;

import javax.swing.AbstractAction;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JPanel;

import org.virion.jam.html.SimpleLinkListener;
import org.virion.jam.util.SimpleListener;

import com.biomatters.geneious.publicapi.components.Dialogs;
import com.biomatters.geneious.publicapi.components.GEditorPane;
import com.biomatters.geneious.publicapi.plugin.Options;
import com.biomatters.geneious.publicapi.utilities.IconUtilities;
```

Next, make the following additions inside the `PhobosOptions` class:

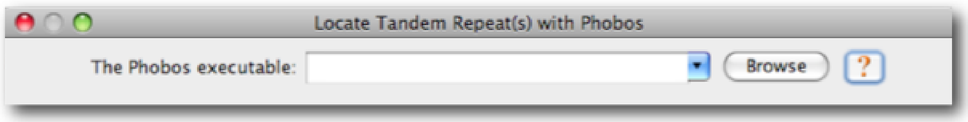
```
public String verifyOptionsAreValid() {
       String executable = codeLocation.getValue();
       if (executable.trim().length() == 0) {
              return "Phobos executable location not set." +
                     "<br><br>" + getExecutableLocationHelp();
       }
       else return null;
}
private String getExecutableLocationHelp() {
       return "This plugin makes use of the external program Phobos by Christoph Mayer to
find tandem repeats.<br><br>The Phobos package can be downloaded from <a href=\"" + PHOBOS_URL
 +  "\">" +  PHOBOS_URL + "</a> " + "and is free for non-commercial use." + "<br><br>After
downloading and extracting the package, point Geneious to the location of the executable.";
}

private JComponent getHelpButtonForLocation() {
       return new JButton(new AbstractAction("",
                     IconUtilities.getIcons("help16.png").getIcon16()) {
```

```
                public void actionPerformed(ActionEvent e) {
                        showHelpForPhobosLocation();
                }
        });
}
private void showHelpForPhobosLocation() {
        Dialogs.showMessageDialog(getExecutableLocationHelp(), "Where to get Phobos");
}
```

You also need to make some changes to the `codeLocationOptions()` method to include the new help button:

```
private void codeLocationOptions() {
        beginAlignHorizontally(null, false);
        codeLocation = addFileSelectionOption("EXE", "The Phobos executable:", "");
        codeLocation.setRestoreDefaultApplies(false);
        addCustomComponent(getHelpButtonForLocation());
        endAlignHorizontally();
        addDivider(""); // Add some empty space to separate this option a bit
}
```

Now, the top of your options dialogue will have gained a 'Help' button next to the 'Browse' button giving useful information about Phobos and where to get it:



Also, the plugin will now check whether the Phobos executable is undefined and if not will also pop up a dialogue explaining where to get it.

It is also possible that the Phobos executable isn't the right one, or the binary that the user points the plugin at isn't Phobos at all so the plugin should verify that it is working correctly and flag that error too. Modify `PhobosExecutionOutputListener.java` to incorporate the following code:

```
private boolean phobosFound = false;
boolean phobosFoundFlag() {
        return phobosFound;
}

public void stdoutWritten(String output) {
        if (output.startsWith("#")) {
                if (output.contains("Phobos")) {
                        phobosFound = true;
                }
        }
```

If Phobos has correctly recognised the command line options passed to it, it will return output on `stdout` so you can use this fact to determine that Phobos has run. A quirk we're using here is that previous versions of Phobos (prior to 3.3.10) don't recognise all the options passed so will only output to `stderr` causing this test to fail and the plugin will report that you need Phobos 3.3.10 to run.

You will also need to change `PhobosAnnotationGenerator.java` to use this information:

```
        if (exec.wasKilledByGeneious()) {
                return null;
        }

        // If the program failed to run correctly or at all throw IOException
        if (!outputListener.phobosFoundFlag()) {
                throw new IOException(" Phobos 3.3.10 or later required");
        }
        resultsList.add(outputListener.getResults());
} catch (IOException e) {
```

Congratulations on getting this far. You have now walked through the evolution of the Phobos plugin and the final code you have is exactly what produces the finished actual Phobos plugin provided to our users. It is now up to you to find other similar projects which you can base on the knowledge you have gained throughout this document.