.NET Bio Code Contribution Guide

Version 1.0 - November, 2011

Abstract

This document describes how to contribute code to the .NET Bio open source project.

For updates to this document and the rest of the .NET Bio documentation, see   
<http://bio.codeplex.com/documentation>

For updates to .NET Bio, see [http://bio.codeplex.com](http://bio.codeplex.com/)

Contents

[Introduction to the .NET Bio contribution process 2](#_Toc311209610)

[Register at CodePlex and accept Terms of Contribution 2](#_Toc311209611)

[Review the C# coding and documentation guidelines 4](#_Toc311209612)

[Review existing code 5](#_Toc311209613)

[Contact .NET Bio coordinators for contribution rights 5](#_Toc311209614)

[Follow on-boarding instructions 5](#_Toc311209615)

[Create a new CodePlex work item 6](#_Toc311209616)

[Write the new code 6](#_Toc311209617)

[Write documentation for the code 6](#_Toc311209618)

[Write unit tests for the new code 7](#_Toc311209619)

[Prepare a Shelveset for code review 8](#_Toc311209620)

[Using RocketSVN for Visual Studio 8](#_Toc311209621)

[Download and Install RocketSVN 9](#_Toc311209622)

[Creating a Patch 9](#_Toc311209623)

[Submit your changes on CodePlex 12](#_Toc311209624)

Disclaimer: This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2010 Microsoft Corporation. All rights reserved.

Microsoft and Windows are trademarks of the Microsoft group of companies.  All other trademarks are property of their respective owners.

Introduction to the .NET Bio contribution process

.NET Bio is a language-neutral bioinformatics toolkit, built as an extension to the Microsoft .NET framework. .NET Bio implements:

* A range of parsers for common bioinformatics file formats.
* A range of algorithms for manipulating DNA, RNA, and protein sequences.
* A set of connectors to biological Web services such as NCBI BLAST.

.NET Bio is available under an open source license. Executables, source code, demo applications, and documentation are freely downloadable from [http//bio.codeplex.com](http://bio.codeplex.com/).

## About .NET Bio code contributions

.NET Bio is open to code contributions from the community (anyone with a valid CodePlex ID), with the goal of extending the range of available functionality to researchers and life scientists everywhere. If you need functions that are not in the basic library, you can implement them easily, in a way that works with the existing functions.

We encourage developers who extend .NET Bio to contribute their code back to the project as open source, so that the community as a whole can benefit from their work. Microsoft researchers are already using .NET Bio in their research, as are an increasing number of academic partners. These researchers will be making code contributions to extend the range and power of .NET Bio, and we encourage you to do the same.

## The .NET Bio contribution process

This document describes the process for you to follow when contributing code to .NET Bio. The process consists of the following steps:

1. [Register at CodePlex and accept Terms of Contribution](#_Toc311208154)
2. [Review the C# coding and documentation guidelines](#_Toc311208155)
3. [Review existing code](#_Toc311208156)
4. [Contact .NET Bio coordinators for contribution rights](#_Toc311208157)
5. [Follow on-boarding instructions](#_Toc311208158)
6. [Create a new CodePlex work item](#_Toc311208159)
7. [Write the new code](#_Toc311208160)
8. [Write documentation for the code](#_Toc311208161)
9. [Write unit tests for the new code](#_Toc311208162)
10. [Prepare a Shelveset for code review](#_Toc311208163)
11. [Request code review from .NET Bio coordinators](#_Toc311208164)
12. [Respond to code review recommendations](#_Toc311208165)
13. [Submit updated shelveset for final review](#_Toc311208166)
14. [Submit changes to the code base](#_Toc311208167)

# Register at CodePlex and accept Terms of Contribution

You must first sign up for an account on CodePlex. Once registered, you can create projects, enter work items, and participate in discussion forums.

To register on CodePlex

* Follow the instructions at <https://www.codeplex.com/site/register>
* Once you are registered, your first stop is [http://bio.codeplex.com](http://bio.codeplex.com/)

## Get and Sign the Outercurve Contribution License Agreement

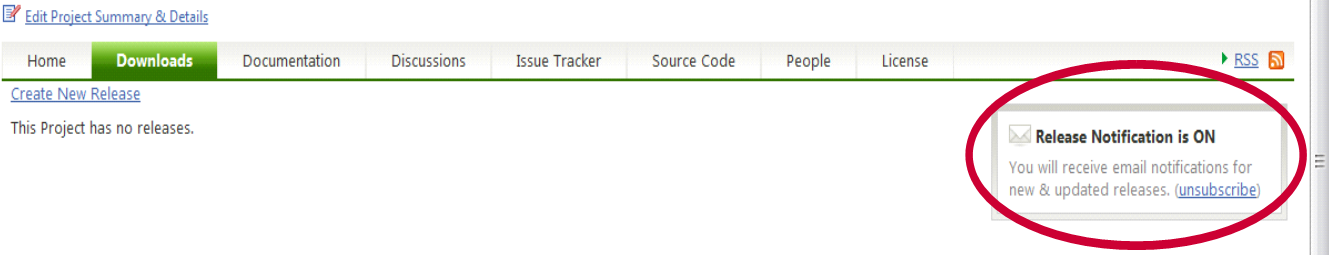
In order to contribute “significant” (a non bug fix of more than 200 lines of code) to the project you must first sign an [Outercurve contribution license agreement](http://www.outercurve.org/About/DevelopmentPractices#agreements). You can get this by contacting one of the project coordinators listed under the people tab of the CodePlex project. You must supply your codeplex user ID and email name with that request so you can be contacted. You will then be sent a copy of the contribution license agreement to sign.

Important: Please read this information carefully to ensure you understand the terms of the agreement.

## Sign up for .NET Bio email and discussion groups

You should ensure that you are signed up to receive email notifications for the following:

* Downloads: Release Notifications
* Discussions: Discussion Notifications
* Issue Tracker: Issue Notifications

  
 Setting Release Notifications Choices on CodePlex Projects

Note: You can choose to receive an update on every change or daily summaries, and you can choose RSS feeds instead of email. The important thing is to stay on top of who is doing what on the project, so that you know who you might need to coordinate with.

You can review and manage your email notification settings in your CodePlex profile as well. There might be other CodePlex project notifications you will want to join. This guide focuses on the .NET Bio groups that you should be aware of.

## Dependent software products

Important: To compile the .NET Bio project, Visual Studio 2010 is required.

For all contributors. In order to write, compile and execute the test components, you will need Visual Studio 2010 Professional addition as we utilize the - Visual Studio Unit Test Framework.

Other optional components:

* For the MBF Sequence Assembler sample application:   
  Microsoft Silverlight 4 SDK  
  <http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=18149>
* For the IronPython scripts:  
  IronPython 2.0.1 Runtime  
  <http://www.codeplex.com/IronPython>
* For building Trident activities and workflows:  
  Trident Version 1.0 or later  
  <http://research.microsoft.com/en-us/collaboration/tools/trident.aspx>
* To automatically generate a help file for the APIs:   
  **Sandcastle**  
  <http://www.sandcastledocs.com>
* To check for errors in .NET managed assemblies:  
  **FxCop**  
  <http://www.microsoft.com/download/en/details.aspx?id=6544>

For academic researchers. If you don’t have retail versions of the required software you can receive free Microsoft Software. To download and install the latest version of Microsoft Visual Studio® (VS2010), visit the Microsoft [DreamSpark](https://www.dreamspark.com/) web site at <https://www.dreamspark.com>.

You will also find other optional components on DreamSpark that you might like to use in your work.

# Review the C# coding and documentation guidelines

When developing code for .NET Bio, follow the guidelines in the following documents:

“.NET Bio C# Coding Conventions”

Presents guidelines for developing code on and for .NET Bio. One of the key components to delivering on the .NET Bio promise of higher productivity is the ability to provide a consistent approach to the programming model and stylistic conventions used throughout .NET Bio development.

“How to Write Documentation Comments in Source Code”

Discusses how to write good documentation comments, specifically for projects that will produce an API reference directly from code comments. The quality bar is high for such projects, because there is little opportunity to modify the text after it is harvested.

“.NET Bio Contributors Template”

Provides a template for a User/Programming Guide for .NET Bio contributions. The format is flexible, so you should feel free to adapt this template to the particular needs of your contribution.

All .NET Bio documentation can be found at <http://bio.codeplex.com/documentation>

# Review existing code

As a participant in an open source project, you can evaluate the existing code first hand. Although most of the detailed technical information is available in the SDK documentation, you might find it easier to simply examine the code files and associated comments. If you use code editing software with reference/use tools such as Visual Studio, it is also helpful to traverse through the code via reference.

One other technique that we highly recommend is to build the samples provided in \source\tools\\* and run them under the debugger. The bio.sln solution file will include all samples and utilities so start with that one. By setting breakpoints and stepping through actively running code, you will be able to see what the order of operation is (at least for a particular sample). Also, by observing the data register during the debugging session, you can follow along with how the data is represented and modified throughout the sequence of execution.

Finally, by taking a look at what others have provided as code changes to the base framework, you can get an idea of what type of code is expected in a contribution. Changelists and ShelveSets can be quite interesting to evaluate, and will provide context for both the new and old code and the related differences. As you become more familiar with the code and contribute to the project, you might have the opportunity to become a code reviewer, in addition to your role as a contributor.

# Contact .NET Bio coordinators for contribution rights

To add code to .NET Bio, you must grant rights to the Outercurve Foundation regarding ownership and copyright. This is an open source project under the OSI Apache 2.0 license.

To be able to contribute

* Send an e-mail to [bio@outercurve.org](mailto:bio@outercurve.org)

Include these details:

Your level of interest in contributing to NET Bio.

Your background and/or experience in programming and bioinformatics.

Your CodePlex User ID and email name so you can accept the Outercurve Terms of Contribution, found at <http://www.outercurve.org/About/DevelopmentPractices#agreements>

# Follow on-boarding instructions

After you have signed the contribution license agreement, you can get started with .NET Bio by following the guidelines in the .NET Bio Onboarding document, found at <http://bio.codeplex.com/documentation>.

The Onboarding document describes how to:

* Get ‘wired’ into the .NET Bio team by joining discussion lists and email notifications, locating relevant web sites, and browsing the documentation.
* Install the necessary tools.
* Create and configure your development environment.

# Create a new CodePlex work item

The .NET Bio community uses the CodePlex Issue Tracker to create and track issues such as feature work, product issues, and tasks.

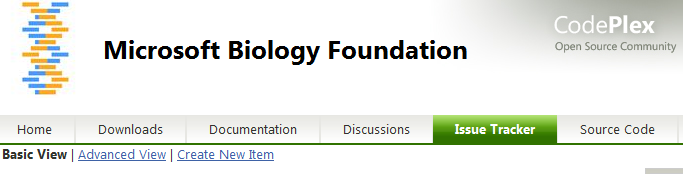
For general help on the Issue Tracker, see [the CodePlex Information and Discussion wiki](http://codeplex.codeplex.com/wikipage?title=Issue%20Tracker&referringTitle=CodePlex%20Documentation).

To create a work item

1. Sign in to CodePlex and go to the .NET Bio project.

2. Click Issue Tracker.

3. Click Create New Item.



4. Set Status to Proposed.

5. Set Type to Task.

6. Enter a clear title and description for your work item.

7. Fill out the remainder of the form as appropriate and click Save.

# Write the new code

Write your new .NET Bio code, following the [.NET Bio coding and commenting guidelines](#_Review_the_C). Build the code as described in the [.NET Bio Onboarding document](#_Follow_on-boarding_instructions).

Important: When you finish coding, run FxCop and correct any errors it finds. FxCop analyzes programming elements in managed assemblies by using rules that return informational messages when the rules are violated. Messages identify any relevant programming and design issues and, when it is possible, supply information about how to fix the issues.

# Write documentation for the code

If your .NET Bio code includes new features for users, then fill out the .NET Bio Contributors Template, found at <http://bio.codeplex.com/documentation>.

If new .NET Bio code includes new APIs for developers, then be sure to:

* Comment the APIs by following the guidelines in “How to Write Code Comments In Source Code” document.
* Use the Sandcastle tool to automatically generate a help file for the APIs.

# Write unit tests for the new code

All code contributions to .NET Bio require unit testing as described in the [.NET Bio Onboarding document](#_Follow_on-boarding_instructions).

Here are some recommendations for creating new unit tests:

* Identify all the public APIs—methods, constructor, properties, or others—available in the framework.
* Create a “happy path” (the expected execution path) test case for each public API and property.
* Write test cases to cover a specific scenario, where one or more public APIs are used.
* Cover as many scenarios as possible.
* Make sure that every line of code is hit to get the maximum code coverage.
* Automate the test cases based on the priority— for example, BVTs, Priority 1 and Priority 2.

## Unit test case generation

namespace Bio.Test | Bio.Test.dll

* Identify all the public APIs—methods, constructor, properties, or others—available in the framework.
* Create a “happy path” test case for each public API and property.
* Write test cases to cover a specific scenario, where one or more public APIs are involved.

For example, the FastA parser have many different override public APIs for parsing. .NET Bio includes many individual public API test cases, such as TestFastaWhenParsingOneOfMany().

An example scenario would be to pass an input FastA file to the parser and get the output sequence object. Then, pass those output sequence objects into FastA formatter, where it produces an output file, which is exactly similar to the original file.

Minimal negative cases are written as part of unit test cases. For example, BadContent() is a negative FastA parser test case, to check bad content in FastA file format.

## Automation test case generation

namespace Bio.TestAutomation | Bio.TestAutomation.dll

* Identify a work item/feature for automation.
* Identify all the public APIs—methods, constructor, properties. and others—available in that feature.
* Come up with all Positive, Negative and Boundary test cases and document the same in the Product Studio/Excel work sheet.
* Identify the priority for all the test cases—for example, BVT, Priority 1 and Priority 2.

As an example, consider the FastA Parser API, which uses a Parse(file-path) method exposed as public. We would test these scenarios for that API as follows:

1. Parse FastA file that has one line sequence
2. Parse FastA file of size less than 35 K (small size)
3. Parse FastA file of size more than 35 K and less than 100 K (medium size)
4. Parse FastA file of size more than 100 K and less than 350 K (large size)
5. Parse FastA file of size more than 350 K (very large size)
6. Parse FastA file that contains only DNA sequence
7. Parse FastA file that contains only RNA sequence
8. Parse FastA file that contains only Protein sequence
9. Parse FastA file that contains DNA and RNA sequence
10. Parse FastA file that contains DNA and Protein sequence
11. Parse FastA file that contains Protein and RNA sequence
12. Parse FastA file that contains DNA, RNA and Protein sequence
13. Parse FastA file that contains Ambiguous DNA sequence
14. Parse FastA file that contains Ambiguous RNA sequence
15. Parse FastA file that contains Ambiguous Protein sequence
16. Parse FastA file with valid file path
17. Parse FastA file with invalid file path
18. Parse FastA file with space in file path
19. Parse FastA file with Unicode characters in file path
20. Parse FastA file with Unicode characters in the FastA file
21. Parse FastA file with file in network location
22. Parse GenBank file instead of FastA
23. Parse GFF file instead of FastA
24. Parse SAM file instead of FastA
25. Parse FastA file that is empty
26. Parse FastA file that is too big for the parser to handle

# Prepare a Shelveset for code review

Use the Shelve command in TFS to store your pending changes—together with pending check-in notes, a comment, and a list of associated work items—on the Team Foundation Server without actually checking them into the source control server.

The output of the Shelve command is known as a shelveset. Give the shelveset a descriptive name and update your CodePlex work item with the name of the shelveset.

For more information, see [Shelve Command in the VS Team System online help](http://msdn.microsoft.com/en-us/library/w6y8ezzs(VS.80).aspx).

# Using RocketSVN for Visual Studio

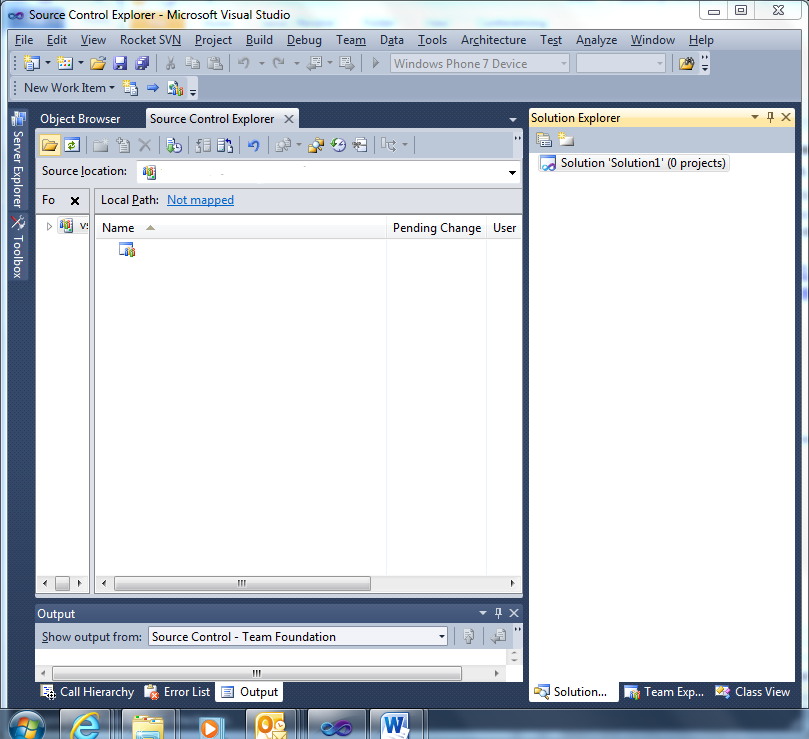
RocketSVN for Visual Studio enables you to commit or update a change inside the Visual Studio IDE. It integrates in both Visual Studio 2008 and Visual Studio 2010.

RocketSVN includes the following:

* Loading Project Files
* Comparing Versions
* Viewing Comments
* Viewing Pending Changes
* Commiting Changes
* How Easy it is to Use

# Download and Install RocketSVN

You can download Rocket at <http://www.axosoft.com/rocketsvn>.



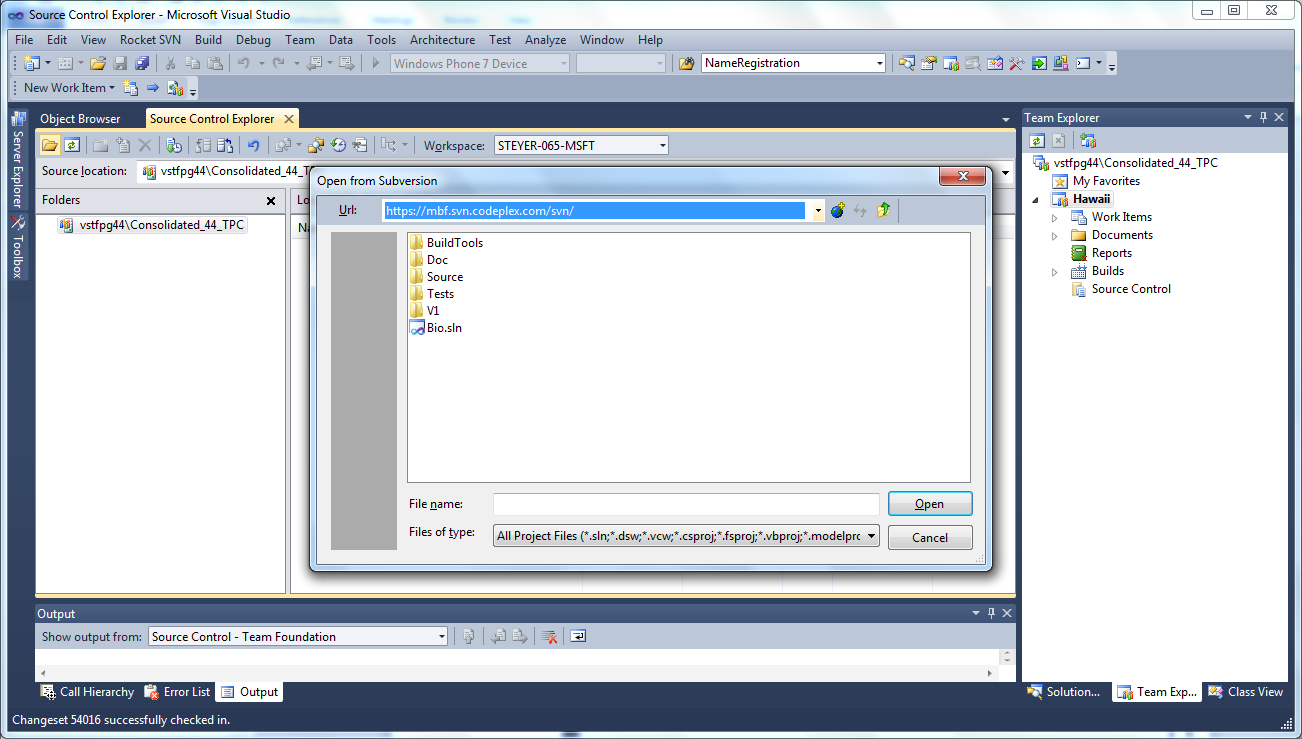
# Creating a Patch

When you are satisfied with your code and all tests have passed prepare your code for submission.

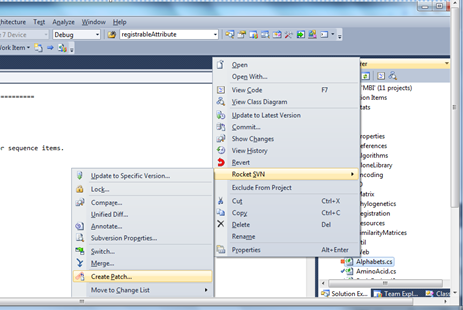
For open source projects the change is managed by submitting a patch file to the project committers, who have write access to the project repository. They review the patch, and then either submit it to the repository or reject it back to the author.

Rocket is integrated with Visual Studio as a Tab on the toolbar.

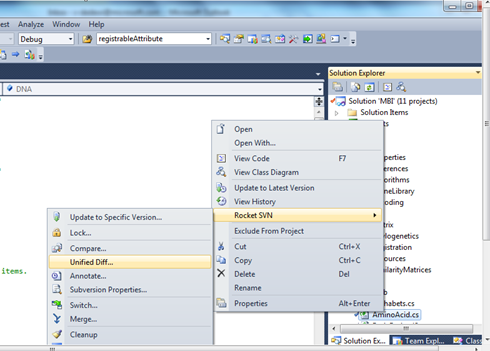
1. Open the Bio.sln solution using the RocketSVN tab.   
   In the **URL of repository** field enter the Subversion URL for the .net Bio project, <https://bio.svn.codeplex.com/svn/>.



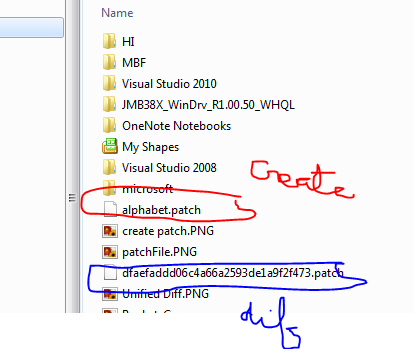
1. Use Visual Studio to edit a file such as alphabet.cs.
2. Save but do not commit the file.
3. Right click on the edited file in Solution Explorer, point to RocketSVN select create patch



You can also use the Unified Diff selection to create a patch file with the .patch extension.



The resulting files from the two processes are shown below:

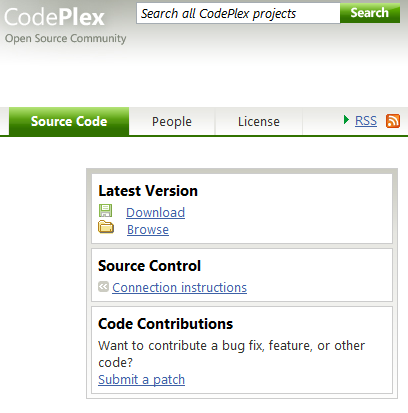


# Submit your changes on CodePlex

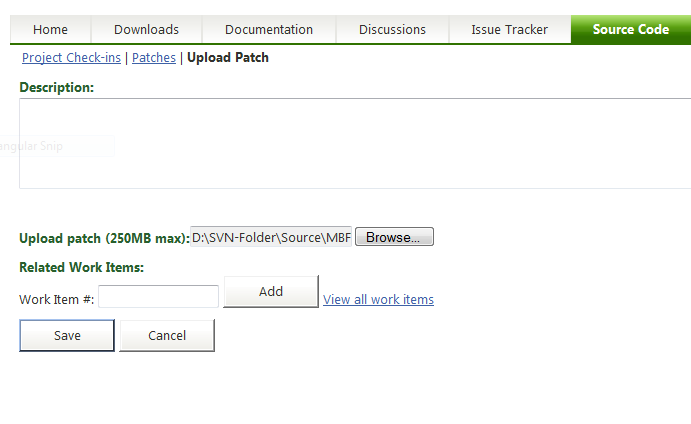
You can submit your new code on CodePlex using **Submit Patches**.

To submit your changes

1. Click Submit a patch on the [.NET Bio CodePlex Source Code](http://bio.codeplex.com/SourceControl/list/changesets) page.



1. On the **Upload a Patch** pageuse the **Browse** button to navigate to the patch and select it. Use the **Add** button to include other files such as unit tests.



1. Click the **Save** button and you are done. Let the Process begin.

## Processing a submittal

Project coordinators will review your submission. The time for review varies, depending upon where the project team is in its schedule. If the team is close to a milestone release, then your contribution might be postponed until the next milestone release. The general guideline for processing a submittal is:

1. Contributor submits a Patch via CodePlex.
2. Committers get automatic notification via email.
3. Committers assign Patch adoption to a specific committer within 48hrs, at which point the assigned committer contacts the contributor via CodePlex to let them know we are working on it. They may also request additional information such as unit tests, test results and test data.
4. The assigned committer responds with code review comments within72 hours of submission of all needed materials including documented code, unit tests, test results and test data.
5. If changes are required, the contributor is notified via email and the submittal process is repeated.
6. The contribution is accepted or rejected.
   1. If the contribution is accepted, the assigned committer merges the contributor's change into the Main development branch within a week of acceptance.
   2. If the contribution is rejected, the assigned committer contacts the contributor with denial reasons.