|  |  |
| --- | --- |
|  | .NET Bio Framework Programming Guide  Version 1.1 June 2013 |

Abstract

The .NET Bio Framework is an open source, reusable .NET Framework library and application programming interface (API) for bioinformatics research. This +document describes the basics of how to implement .NET Bio applications.

.NET Bio Framework software and documentation are available at: <http://bio.codeplex.com/>.

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.

© 2011-2013 The Outercurve Foundation.

Distributed under Creative Commons Attribution 3.0 Unported License.

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

Contents

[Introduction 4](#_Toc294553062)

[Terminology 4](#_Toc294553063)

[Getting Started 7](#_Toc294553064)

[Installation 7](#_Toc294553065)

[Prerequisites 8](#_Toc294553066)

[Hardware and Software Requirements 8](#_Toc294553067)

[Start a New .NET Bio Framework Project 10](#_Toc294553068)

[.NET BioWorkflow Activities for Project Trident 13](#_Toc294553069)

[Scenarios 13](#_Toc294553070)

[Scenario 1 14](#_Toc294553071)

[Scenario 2 14](#_Toc294553072)

[Scenario 3 15](#_Toc294553073)

[How-To’s 16](#_Toc294553074)

[What’s New 17](#_Toc294553075)

[A .NET Bio Framework Quick Start 19](#_Toc294553076)

[How to Align Sequences - AlignSequences Sample Application 21](#_Toc294553077)

[Migrating the AlignSequences Example from MBF to .NET Bio 23](#_Toc294553078)

[AlignSequences Notes 25](#_Toc294553079)

[.NET Bio Framework Architecture 26](#_Toc294553080)

[Sample Applications and Utilities 27](#_Toc294553081)

[I/O and Analysis 28](#_Toc294553082)

[Object Model 30](#_Toc294553083)

[Input and Output: Parsers and Formatters 30](#_Toc294553084)

[Parsers 32](#_Toc294553085)

[Formatters 34](#_Toc294553086)

[Input and Output: Web Service Connectors 34](#_Toc294553087)

[Object Model: Sequences and Related Types 35](#_Toc294553088)

[Alphabets 36](#_Toc294553089)

[The Sequence Object 37](#_Toc294553090)

[Sequence Manipulation 40](#_Toc294553091)

[The SequenceRange Object 41](#_Toc294553092)

[The AlignedSequence Object 41](#_Toc294553093)

[Object Model: Other Types 42](#_Toc294553094)

[Phylogenetics 42](#_Toc294553095)

[SNP Items 42](#_Toc294553096)

[Bio.Matrices 43](#_Toc294553097)

[Data Processing: Algorithms 44](#_Toc294553098)

[Example: How to Manipulate a Sequence 46](#_Toc294553099)

[Migrating the SequenceManipulation example from MBF to .NET Bio 47](#_Toc294553100)

[SequenceManipulation Notes 49](#_Toc294553101)

[Example: How to Submit a .NET Bio Framework Web Services Request 50](#_Toc294553102)

[BlastRequest Sample 51](#_Toc294553103)

[BlastRequest Notes 52](#_Toc294553104)

[Extending .NET Bio Framework: How to Register Add-in Components 55](#_Toc294553105)

[Resources 57](#_Toc294553106)

[Appendix A: Sample GenBank Data File 59](#_Toc294553107)

[GenBankSample1.gbk File 59](#_Toc294553108)

[GenBankSample2.gbk File 60](#_Toc294553109)

[Appendix B: RNA and Protein Alphabets 62](#_Toc294553110)

[The RNA Alphabet 62](#_Toc294553111)

[Protein Alphabet 63](#_Toc294553112)

# Introduction

The .NET Bio Framework is an open source, reusable .NET Framework library and application programming interface (API) for bioinformatics research. Application developers can use .NET Bio Framework to perform a wide range of tasks, including:

* Import DNA, RNA, or protein sequences from files with a variety of standard data formats, including FASTA, FASTQ, GenBank, GFF, and BED.

This document focuses on DNA sequences, but you use similar procedures for the other sequence types.

* Construct sequences from scratch.
* Manipulate sequences in various ways, such as adding or removing elements or generating a complement.
* Analyze sequences using algorithms such as Smith-Waterman and Needleman-Wunsch.
* Submit sequence data to remote Web sites—such as a Basic Local Alignment Search Tool (BLAST) Web site—for analysis.
* Output sequence data in any supported file format, regardless of the input format.

The project represents sequence data and metadata with format-independent **Sequence** objects. These objects efficiently store sequence data in a variety of encoded formats and provide a flexible and robust way to represent sequences in the project environment.

The project’s applications can be implemented in a variety of languages, including C#, F#, Visual Basic® .NET, and IronPython. You can also work with sequences using an add-in for Microsoft Office Excel and you can build Silverlight applications using bio.Silverlight.dll. For details, see “.NET Bio Extension for Excel User’s Guide,” listed in “[Resources](#_Resources)” at the end of this document.

This document describes the basics of how to implement project applications in C#; other languages follow a very similar programming pattern.

# Terminology

This section defines some basic bioinformatics terminology that is relevant to the project. It contains only terms that are used later in this paper; it is not a complete list.

Assembler

Sequencer assembler algorithms used to merge short sequences or reads to reconstruct an original or base sequence.

Annotation

The process of attaching biological information to sequences. It encompasses identifying elements on the genome, a process called gene prediction, and attaching biological information to these elements.

BAM

A binary equivalent to SAM.

BED

Browser Extensible Display. A plain text file format for data that describes sequence ranges.

Bioinformatics

A discipline that uses mathematical, statistical and computational approaches to analyze DNA and amino acid sequences and related information.

BLAST

The Basic Local Alignment Search Tool (BLAST) compares nucleotide or protein sequences to sequence databases and calculates the statistical significance of matches. BLAST can be used to infer functional and evolutionary relationships between sequences as well as help identify members of gene families.

Breakpoint

The situation where the alignment of a read to the reference consists of more than one contiguous segment, or a single segment that does not extend to the end of the read.

Consensus

A reconstructed sequence of nucleotides or amino acids inferred from an alignment of multiple subsequences. It is also known as a contig.

Contig

A set of nucleotide or amino acid sequences, presumably part of a larger molecule, that have been aligned and overlap with each other.

DNA (deoxyribonucleic acid)

A molecule that consists of a double chain of nucleotides and codes the genetic information for all organisms.

EBI (European Bioinformatics Institute)

A bioinformatics research institute. It hosts one of the available BLAST services.

FASTA

FASTA format—also known as Pearson format—is a text-based data format for representing nucleotide or peptide sequences. It represents base pairs or amino acids with single-letter codes and allows the sequences to be preceded by sequence names and comments.

FASTQ

A plain text format for storing sequence data that combines a FastA sequence with its quality data.

GFF (general feature format)

A plain text file format for describing DNA, RNA, and protein sequences.

GenBank

The GenBank sequence database is an annotated open-access, collection of all publicly available nucleotide sequences and their protein translations. It is hosted by the NCBI as part of the International Nucleotide Sequence Database Collaboration (INSDC).

Genomics

The study of genetic sequences.

homologues

Same copy of DNA. Such as in the case of two copies for all autosomal chromosomes, one coming from the mother and the other coming from the father. The pair is called homologues.

k-mer

Identifies a region within molecules such as DNA.

NCBI

The National Center for Biotechnology Information.

nucleotide

The basic structural unit of DNA and RNA. They are usually referred to by their purine base. DNA uses four nucleotides: adenine, guanine, thymine, and cytosine, commonly abbreviated as A, G, T, and C. RNA also uses A, G, and C, but replaces T with uracil (U).

Phylogenetics

A phylogenetic tree describes evolutionary relationships between organisms that derive from a common ancestor.

Polyploidy

Occurs in cells and organisms when there are more than two paired (homologous) sets of chromosomes.

Protein

A molecule that consists of a chain of amino acids.

RNA (ribonucleic acids)

A single chain of nucleotides.

Sequence

Defines the structure of polymers such as DNA, RNA, and proteins.

SAM (sequence alignment map)

A plain text file format for data that describes nucleotide alignment.

Scaffold

A non-redundant sequence formed by joining one or more contig sequences.

Shotgun sequencing

Also known as shotgun cloning, a method used for sequencing long DNA strands. DNA is broken up randomly into numerous small segments, which are sequenced using the chain termination method to obtain reads. Multiple overlapping reads for the target DNA are obtained by performing several rounds of this fragmentation and sequencing.

SNP (single-nucleotide polymorphism)

Items represent sequence variations between species or paired chromosomes.

Synteny

The condition of two or more genes being on the same chromosome whether or not there is demonstrable linkage between them.

# Getting Started

This section describes basic system requirements and installation, and summarizes steps for starting a .NET Bio project and building it.

References and software described in this discussion are summarized in “[Resources](#_Resources)” at the end of this paper.

## Installation

Application developers have two primary installation options from based on whether you are participating in project as a contributor or a committer (the “Overview” document describes the roles). The essential difference is that contributors access latest deployed project code on [http://bio.codeplex.com/](http://biodex.codeplex.com/) and committers have Partner Credentials and access the active codebase.

For details on how to become a contributor and a committer, download the “Contributor Guide,” “Becoming a Committer” and “Committers Guide” documents from the site’s **Documentation** tab. The project is hosted on <http://bio.codeplex.com/>.

* Contributors download the CodePlex source code.

This option allows you to use and modify the .NET Bio Framework source licensed under Apache 2.0. You have access to the deployed changes and can contribute code to the project. You can build all DLLs by loading Bio.sln and running **Build Solution**.

- OR -

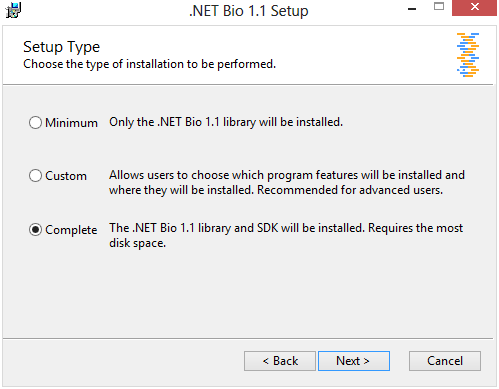
* Committers synchronize to the .NET Bio Framework source tree in the active repository. This option requires Partner Credentials (see the “Becoming a Committer” document) and provides access to the latest changes. You can also contribute code directly to the project. The Framework source tree is a single Visual Studio® 2010 solution, so you can build all Framework DLLs by loading Bio.sln and running **Build Solution**.

The complete installation option installs everything that you need to implement the project’s applications—including all Bio DLLs—under Program Files\.NET Bio\1.0\Framework or Program Files (x86)\.NET Bio\1.0\Framework, on x86 and x64 systems, respectively.

You can also just run the Framework installer, Bio.msi, (select the **Complete** install option to install the software development kit (SDK)) provided on Codeplex. This option installs everything that you need to implement Framework applications—including all Framework DLLs—under the Program Files\.NET Bio Framework directory. However this option provides the project libraries and not the source code so you cannot modify the underlying source code.

All options include documentation and samples. For more details, see the [.NET Bio CodePlex Web site](http://bio.codeplex.com/).

Choose the **Complete** install option on the installation **Setup Type** page when installing the .NET Bio Framework package. The **Complete** install provides the SDK which includes the Bio Console Application template.



You can also download and install the Excel Add-in and the Sequence Assembler tool after running the .NET Bio Framework installer. For more details and download instructions, see the .NET Bio home page on CodePlex.

## Prerequisites

This document assumes that you have at least:

* Basic programming skills.
* Familiarity with using Microsoft Visual Studio® to program .NET applications with C#.
* Basic understanding of programming for Web services.

## Hardware and Software Requirements

You must have the following hardware with the following software installed in order to implement project applications:

### Hardware Requirements:

* A computer that can run Visual Studio 2010 or 2012.
* Optionally, a network connection for using Web service methods.

### Software Requirements:

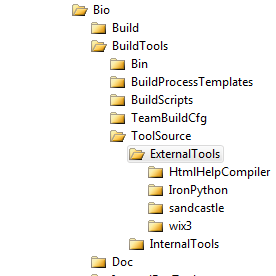
* Windows® 7 or later, x86 or x64 versions
* Visual Studio 2010 or 2012
* .NET Framework 4.0, which is included with Visual Studio
* .NET Bio 1.1 or later

You can install the DLLs or build them yourself from the Framework source code, depending on which installation option you choose.

### Optional software includes:

|  |  |
| --- | --- |
| Optional Component | Description |
| IronPython 2.7 Runtime <http://www.codeplex.com/IronPython> | Used for the IronPython scripts, if you want to use this language to implement project applications. |
| Trident Version 1.0 or later <http://tridentworkflow.codeplex.com> | Used for building Trident activities and workflows. |
| **Sandcastle** and  **Sandcastle Help File Builder** <http://shfb.codeplex.com/>/ | Used to automatically generate a help file for the APIs.  You must use the June 2010 or later releases of these applications to build the Bio API reference. |
| **VSTest** | Used for creating and running unit test cases.  For more information on Visual Studio 2010 testing see [Testing the Application](http://msdn.microsoft.com/en-us/library/ms182409.aspx) on MSDN. |
| **FxCop** <http://www.microsoft.com/downloads/en/details.aspx?displaylang=en&FamilyID=917023f6-d5b7-41bb-bbc0-411a7d66cf3c> | To check for possible design, localization, performance, and security improvements in .NET managed assemblies |
| **WIX** | Used for building the setup installer. |
| Microsoft HPC Pack 2008 R2 Client Utilities Redistributable Package with Service Pack 1 (<http://www.microsoft.com/downloads/en/details.aspx?FamilyID=0a7ba619-fe0e-4e71-82c8-ab4f19c149ad>) | Used to build the HPC portion of the library – this is an optional component and not built when you use the **Bio.Core** solution. |
| Microsoft HPC Pack 2008 R2 SDK  (http://www.microsoft.com/downloads/en/details.aspx?FamilyID=BC671B22-F158-4A5F-828B-7A374B881172) | Used to build the HPC portion of the library – this is an optional component and not built when you use the **Bio.Core** solution. |

**Note**: CreateSetup.cmd has several external tool dependencies in order to create the msi installers. You must download the tools to the locations at $/../Bio/BuildTools/ToolSource/ExternalTools in the source repository.



For more information on these software packages, see “[Resources](#_Resources)” at the end of this document.

## Start a New .NET Bio Framework Project

The Bio API can be used in a variety of .NET application and library types, so the appropriate project template is usually determined by user-interface (UI) requirements and your programming preferences. There are two basic project types: console applications and graphical user interface (GUI) applications. For simplicity, the examples discussed in this document are console applications.

This section describes how to set up both application types.

### Project Console Applications

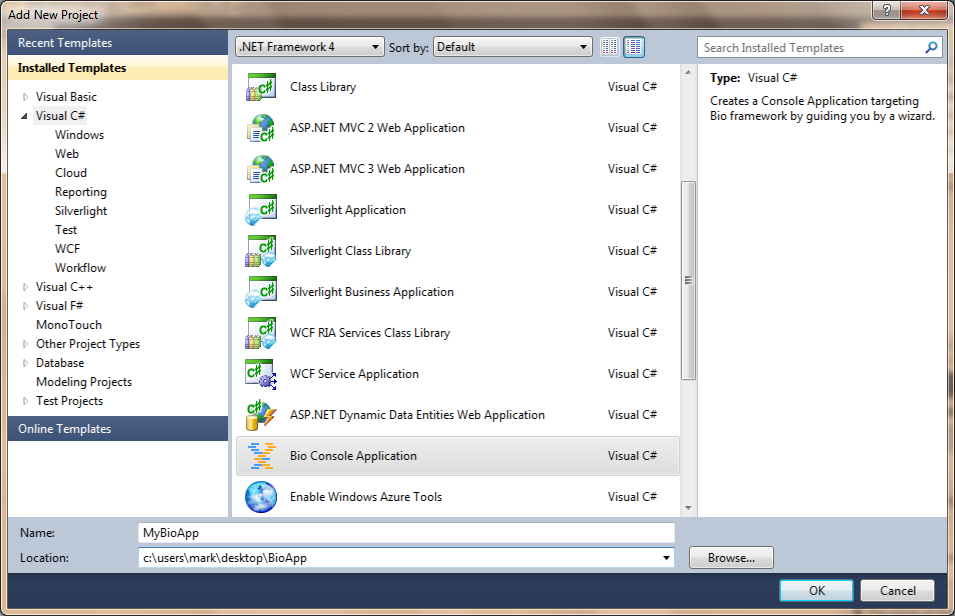
For console applications, the simplest approach is to use the Visual Studio Bio Console Application template, which is installed with the Bio package when you select the **Complete** install option on the installation **Setup Type** page. This template automatically references the appropriate DLLs and provides starting code.

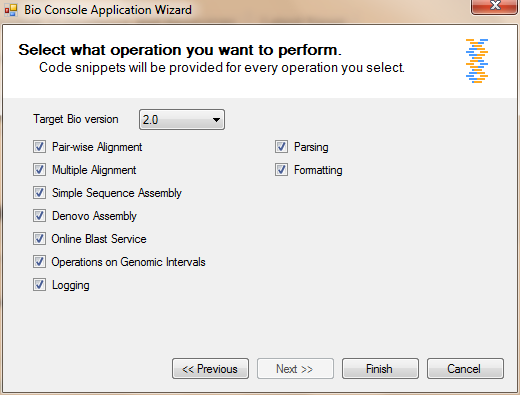
To start a new .NET Bio Framework console application

1. Open the Visual Studio **New Project** dialog box.

To open the dialog box, open the **File** menu and click **New**\**Project**.

2. Select Visual C# in the Installed Templates Pane.

3. Select the **Bio Console Application** template, provide an appropriate name and location, and click **OK** to opens the **Bio Console Application** wizard.  


4. Click **Next**, select the appropriate operations, and click **Finish** to open the new project.  


Visual Studio automatically displays the project’s program.cs file, which contains the template code.

Each of the available operations adds appropriate method templates to program.cs—including any required using directives and references any required DLLs. You can then use the method templates as starting points for your implementation. Bio Console Application includes the following operations:

* Pair-wise Alignment: Creates a method template for aligning two sequences using the Needleman-Wunsch algorithm.
* Multiple Alignment: Creates a method template for aligning multiple sequences using the PAMSAM algorithm.
* Simple Sequence Assembly: Creates a method template for performing simple sequence assembly using the Needleman-Wunsch algorithm for global alignment.
* Denovo Assembly: Creates a method template for performing sequence assembly using the Padena assembler.
* Online Blast Service: Creates several method templates to manage submission of data to a BLAST Web site.
* Operations on Genomic Intervals: Creates a method template for merging two sequence ranges.
* Logging: Creates a method template for writing strings to a log.
* Parsing: Creates a method template for parsing a FastA data file.
* Formatting: Creates a method template for formatting a FastA data file.

Many operations, such as parsing, can be performed by a variety of components. The template selects a particular component—such as the FastA parser for the parsing operation—but you can easily modify the code to use the appropriate components for your application.

1. Add your code to the **Main** method. Call the supplied methods to get, save and manipulate the sequences.

### .NET Bio Framework GUI Applications

Applications that require significant user interaction typically use a GUI, and are usually based on Windows Forms or Windows Presentation Foundation (WPF). There is no Bio template for GUI applications, but the following procedure describes how to set up a standard project for the Framework.

To start a new GUI-based .NET Bio Framework project

1. Create a new Visual Studio project of the appropriate type.

2. Reference the following Bio DLLs:

(Required) Bio.dll, which contains the core framework.

(Optional) Bio.WebServiceHandlers.dll, if you want to use the Framework Web service API.

3. Select the correct .NET target framework. To do this:

Right-click the project name in **Solution Explorer**, and click **Properties**.

In the left pane of the **Properties** window, click **Application**.

Click **.Net Framework 4** in the **Target Framework** dropdown list.

For a full end-to-end WPF application example see the training material, particularly Module 5: Algorithms, at [.NET Bio Training](http://research.microsoft.com/en-US/projects/bio/training.aspx).

## .NET BioWorkflow Activities for Project Trident

Microsoft Project Trident: A Scientific Workflow Workbench is a set of applications—based on the Windows Workflow Foundation (WF)—that provide a framework for constructing and running data analysis schemes. Scientists construct their scheme by using Trident Composer to “snap” together components—called activities—to form a data analysis pipeline—called a workflow. Each activity performs a specific task, and Trident manages the overall flow of control and data through the pipeline.

Trident Workbench can be a flexible and powerful tool for bioinformatics research. Even for scientists with limited programming experience can use the Trident Workbench graphical user-interface to quickly construct and run sophisticated and powerful data analysis workflows. For example, you could use a data input activity to reads the data from a particular format, pass that data to an analysis activity, pass the processed data from the analysis activity to a display activity, and finally pass the processed data to a data storage activity to store the data on the hard drive. If you want to read data with a different format, you can simply snap in a new data input activity.

The ability of Trident Workbench to handle the requirements of a particular line of research depends on availability of suitable activities. However, if the standard set of activities does not meet your project requirements, you can implement custom Trident activities to handle specialized procedures. These activities can then be snapped into a Trident workflow like any other Trident activity.

Trident activities are similar to regular WF activities, so implementing them is straightforward. For details, see “Trident Programming Guide” in the Project Trident download package listed in “[Resources](#_Resources)” at the end of this document.

You can find several examples of Trident bioinformatics activities in the following locations:

1. If you did a complete install of the project, an **SDK** folder is created in the install path under Framework and has examples under the **Samples\TridentWorkflows\Source** folder.
2. If you downloaded the source for .NET Bio Framework then the samples are located under **Source\Tools\Bio.Workflow**.

# Scenarios

The following are examples of scenarios that a researcher or scientists in the bioinformatics community may wish to explore.

## Scenario 1

A researcher has sequenced multiple strains of Mycobacterium tuberculosis, Streptococcus pneumonia and Staphylococcus aurous in order to understand virulence, drug resistance and other phenotypic differences between strains. Since one strain’s sequence is available, they would like to use comparative genome assembly to assist in the sequence assembly of other strains.

## Scenario 2

A scientist wants to pull together the read output of DNA sequencing machines to product a complete, contiguous sequence of bases that represents the genetic content for the sample under review. The sequencing machines break the DNA into small “chunks” or reads that are read by the hardware and the results written to a file.

The scientist must then assemble these reads to produce the DNA sequence for the entire original genome by matching the read overlaps. To reconstruct the original sequence, each position must be sampled multiple times to reduce the likelihood that “holes” are left in the information. This over sampling produces a larger amount of data that must be validated and processed to produce the genome.

### Implementing the Scenario

To accomplish this task the researcher should take the input files, and do two types of assembly:

1. Assemble the sample FastA files using Padena algorithm.
2. Comparative Assembly whereby the provided reference genome is used as a guide in the assembly process.

### Comparative Assembly

The following five major steps of the comparative assembly process are implemented as atomic units for use in any combination or isolation.

1. Read Alignment
   1. Provides increased capacity and performance improvements for the generation of Maximum Unique Matches (MUMs).
   2. Provides sorting using LIS.
   3. Provides increased capacity and performance improvements for the NUCmer implementation.
2. Repeat Resolution: A new feature of the library to optimize comparative assembly by eliminating repeats.
3. Layout Refinement: A new feature of the library to optimize comparative assembly by refining the layout.
4. Consensus Generation: capacity and performance improvements for generation of consensus from aligned reads
5. Scaffolding: Provides capacity and performance improvements to the generation of, storage of, and access to the aligned reads scaffold information.



## Scenario 3

Variant Database: Use Cases

A collection of many genomes, suitably boiled down that will allow you to answer questions such as:

1. Given a particular reference location

* What variants are seen there?
* Which sequences have or don't have each variation (as opposed to unmeasured).
* The read quality/frequency information for each sequence.

1. Same as #1, but for a given range of locations such as chr1:1-100000
2. Find all variations in individual X that are in a coding region or splice site.
3. Given a list of individuals, find all of their homozygous nonsense variations or mis-sense variations
4. Given a list of individuals, find all genes in which all of the individuals have a variation (though not necessarily the same variation).
5. Given a list of individuals, produce a list of genes for which at least one individual had a variation, and which fraction of the individuals contained a variation in that gene (though not necessarily the same variation).

Components

* A set of genomes optimized for positional queries.
* A set of variants and annotations.
* Query tools, e.g. Data Lab.

Process

* **Automated Annotation**: Creating workflows that get data from the variant database, apply tools from the science portal, and put the resulting annotation back into the variant database.
* **Reference Rationalization:** Use the data to help redefine what we mean by the reference, and this updated reference to create better variant calls. Turning the pencil sharpener on itself.
* **Assemblies and Variants:** Enhanced variant calling, including looking for copy number variants, translocations, etc. What’s left in the garbage heap?
* **Visualization**: Beyond the Genome Browser.

**Metadata Grid:** Enable automated generation and propagation of metadata using iRODS. Use this metadata to enhance analysis & data management.

## How-To’s

This document includes the following topics on how to perform a variety of tasks:

Programming Guide How-To’s

|  |  |
| --- | --- |
| How To | Description |
| How to use built in parsers. | See [How to Create a Sequence Object](#_How_to_Create) and [Parsers](#_Parsers). |
| How to use built in formatters. | How to use an existing formatter, see [Formatters](#_Formatters). |
| [How to Create a Sequence Object](#_How_to_Create) | Two examples are shown.   * Use a parser to read data from a file and create a sequence. * Create a sequence from scratch. |
| [How to Enumerate a Sequence](#_How_to_Enumerate) | Enumerate a sequence with **foreach.** |
| [How to Manipulate a Sequence](#_Example:_How_to_1) | The example includes descriptions on the following:   * How to work with sequence fragments * How to load a sequence into memory. * How to write a sequence. * How to save a sequence. * How to generate a complement or reverse sequence. |
| [How to Submit a Framework Web Services Request](#_Example:_How_to) | The example includes descriptions on the following:   * How to use Blast * How to use WebRequest |
| [How to Register Add-in Components](#_Extending_Biodex_Framework:) | Describes the basic registration model. |
| [How to align sequences](#_How_to_Align) | A sample application aligning two sequences.   * How to perform sequence alignment * How to use **SequenceStatistics** to iterate through the sequence. |
| How to use standard bioinformatics algorithms | * sequence alignment * sequence assembly * translation * sequence searching |
| To start a new GUI-based .NET Bio Framework project | See [.NET Bio Framework Gui Applications](#_Bio_Framework_GUI). |
| To start a new .NET Bio Framework console application | See [Project Console Applications](#_Project_Console_Applications). |
| How to migrate from version 1 to version 2 | [Migrating the AlignSequences Example from MBF 1.0 to .NET Bio 2.0](#_Migrating_the_AlignSequences_1)  [Migrating the SequenceManipulation Example from MBF 1.0 to .NET Bio 2.0](#_Migrating_the_SequenceManipulation) |

# Porting from the Microsoft Biology Foundation (MBF)

The .NET Bio Framework historically came out of the Microsoft Biology Foundation (MBF) and Microsoft Biology Tools (MBT). The following API were added, removed or changed in the migration from MBF to .NET Bio Framework:

Change list for .NET Bio Framework version 1.0

|  |  |
| --- | --- |
| Change | Description |
| **FastaParser -> FastAParser** | A parser is tightly bound to one file as the filename can only be provided through the constructor.  Implements **Parse** for on demand access to sequences.  Returns **IEnumerable<ISequence>** instead of **IList<ISequence>** |
| **Sequence** | Constructors take a filename or byte array only.  Works with and returns **bytes** instead of **ISequenceItems**.  **Removed -** editing options **Insert**, **Remove**, **Replace**.  **Removed - IsDataVirtualized**, **MapToAlphabet**, **Blocks**, **PatternFinder**, **VirtualDataProvider.**  **Added -** **GetSubSequence** method. |
| **SequenceCollection** | A new implementation to replace **IList<ISequence>**.  Will be virtualized. Returns a new instance of **Sequence** class on every request.  Provides flags to indicate things such as if the sequence list is fully loaded. |
| **SequenceReader** | New implementation. |
| **IAlphabet** | Derived from **IEnumerable<Byte**> instead of **ICollection<ISequenceItem>**  **Removed - LookupByValue**, **LookupBySymbol**. |
| **DnaAlphabet** | **AddNucleotide** method. |
| **SequenceParsers** | **FindPaserByFilename** - Finds a parser for the specified file and opens the file. |
| **ISequence : IEnumerable<byte>**. | Uses **IEnumerable<byte**> instead of **IList<>**.  **ISequence** – reduced to just an indexer that returns the byte.  **Changed – Complement** and **RevComplement** are now available as **GetComplementedSequence** and **GetReverseComplementedSequence**.  **Removed – IsReadOnly**, sequences are readonly and there is no other way to make them read/write.  **Removed - Encoding**, **Statistics**, **MoleculeType**, Documentation – removed.  **Removed -** Any methods for editing, such as **Replace**, **Insert**, the sequence.  **Removed - Clone**. |
| **IParser** | **Removed - Alphabet** and **Encoding** Properties.  Parsers and formatters no longer take encodings. We removed the whole encoding class. |
| Data Virtualization | **Removed** |

For further information on migrating from version 1 to version 2 see the following sections:

[Migrating the AlignSequences Example from MBF to .NET Bio](#_Migrating_the_AlignSequences_1)

[Migrating the SequenceManipulation Example from MBF to .NET Bio](#_Migrating_the_SequenceManipulation)

# A .NET Bio Framework Quick Start

This section introduces the basics of .NET Bio Framework programming by walking you through a simple console application, AlignSequences, which introduces the basic features of the Framework API and programming model. Subsequent sections describe the Framework in more detail.

Alignment is a methodology for arranging the sequences of DNA, RNA, and proteins to identify the regions of similarity that may be a consequence of functional, structural or evolutionary relationships between the sequences. This project provides for sequence alignment.

AlignSequences uses the following programming pattern, which is used by many Framework applications:



1. Read input sequences from storage and convert them to .NET Bio Framework objects.
2. Validate the data.
3. Display the data and metadata.
4. Manipulate or analyze the sequences.
5. Write the processed sequence data to storage.

Once you have installed the Framework, you can build and run AlignSequences as follows.

To build and run AlignSequences

1. Open Microsoft Visual Studio 2010 and create a new Visual C# console application named AlignSequences.

2. Open Program.cs and replace the contents with the code from Listing 1 in the following section.

3. Add a reference to Bio.dll.

4. Open the project’s **Properties** page and set the **Target Framework** property to “.NET Framework 4.”

To open the **Properties** page, right-click the project in **Solution Explorer** and click **Properties** on the popup menu.

5. Obtain two GenBank data files, as described following this procedure.

6. Build the application.

7. Press CTRL+F5 to run the application.

AlignSequences works with any suitable GenBank files. You can obtain a wide variety of such files from the GenBank Web site (listed in the “[Resources](#_Resources)” section). A convenient example for learning purposes is the *Saccharomyces cerevisiae* gene sample. The GenBank home page includes a link that describes the sample.

For convenience, Appendix A contains example data, with abbreviated sequences. From a programming perspective, they work in much the same way as the complete sequences, but keep the output to a manageable length. You can also use the complete *Saccharomyces cerevisiae* data from the Web page if you prefer.

Use the samples as follows:

* The first sample data set is a truncated version of the *Saccharomyces cerevisiae* sample data.

Copy the data to text editor such as Notepad, and save the file as GenBankSample1.gbk.

* The second sample data set is a modified version of GenBankSample1.gbk. It was created by adding two groups of nucleotides to the beginning of the original sequence and removing two groups from the end. It also replaces a few of the nucleotides with ‘r’, which represents an ambiguous G or A value.

Copy this data with appropriate metadata to a file named GenBankSample2.gbk.

For a link to the *Saccharomyces cerevisiae* sample, see the “[Resources](#_Resources)” section.

**Tip:** To simplify the code, the example assumes that the input data files are in the project output folder with AlignSequences.exe. The easiest approach is to add the data files to the project, select each file in **Solution Explorer**, and set the file’s **Copy to Output Directory** property to “**Copy Always**.”

## How to Align Sequences - AlignSequences Sample Application

Listing 1 is a slightly abbreviated version of the actual sample, as noted in the example. If you prefer, you can add additional **Console.WriteLine** statements to print the data from the second sequence. To do this, just insert a copy of the code for the first sequence, and change *testSequence1* to *testSequence2*. However, the example compiles and runs as-is.

The numbered comments identify the key parts of the code and are discussed in the notes that follow Listing 1.

There have been a number of changes to the code in this version. For details see the [Migrating the AlignSequences Example from MBF to .NET Bio](#_Migrating_the_AlignSequences_1) section in this document.

Listing 1: AlignSequences

//[1]

using System;

using System.Collections.Generic;

using System.Linq;

using Bio;

using Bio.Algorithms.Alignment;

using Bio.IO.FastA;

using Bio.IO.GenBank;

using Bio.SimilarityMatrices;

namespace AllignSequences

{

class AllignSequences

{

static void Main(string[] args)

{

//[2]

GenBankParser parser1 = new GenBankParser();

parser1.Open("GenBankSample1.gbk");

ISequence testSequence1 = parser1.Parse().First();

GenBankParser parser2 = new GenBankParser();

parser2.Open("GenBankSample2.gbk");

ISequence testSequence2 = parser2.Parse().First();

//[3]

DnaAlphabet dna = DnaAlphabet.Instance;

Console.WriteLine("Sequence 1\n");

SequenceStatistics sequenceStatistics1 = new

SequenceStatistics(testSequence1);

foreach (byte item in dna)

{

Console.WriteLine("{0} = {1}", (char)item,

sequenceStatistics1.GetCount(item));

}

Console.WriteLine("\n\n");

//Omitted: Print statistics for the second sequence

//[4]

Console.WriteLine("ID = {0}", testSequence1.ID);

Console.WriteLine("MoleculeType = {0}",

testSequence1.Alphabet.Name);

foreach (byte nuc in testSequence1)

{

Console.Write((char)nuc);

}

//Omitted: Print the data and metadata for the second sequence.

Console.WriteLine("\n\n");

//[5]

SimilarityMatrix simMatrix = new

SimilarityMatrix(

SimilarityMatrix.StandardSimilarityMatrix.Blosum50);

int gapPenalty = -8;

NeedlemanWunschAligner nwAligner = new NeedlemanWunschAligner();

nwAligner.SimilarityMatrix = simMatrix;

nwAligner.GapOpenCost = gapPenalty;

IList<IPairwiseSequenceAlignment> result =

nwAligner.AlignSimple(testSequence1, testSequence2);

foreach (IPairwiseSequenceAlignment item in result)

{

Console.WriteLine("First Sequence: ");

foreach (byte symbol in item.FirstSequence)

{

Console.Write((char)symbol);

}

Console.WriteLine("Second Sequence: ");

foreach (byte symbol in item.SecondSequence)

{

Console.Write((char)symbol);

}

Console.WriteLine("Consensus: ");

foreach (byte symbol in

item.PairwiseAlignedSequences[0].Consensus)

{

Console.Write((char)symbol);

}

}

//[6]

ISequence outputSequence =

result[0].PairwiseAlignedSequences[0].Consensus;

FastAFormatter outputFormatter = new FastAFormatter();

outputFormatter.Open("fasta\_out.fasta");

outputFormatter.Write(outputSequence);

outputFormatter.Close();

}

}

}

## Migrating the AlignSequences Example from MBF to .NET Bio

This section highlights the changes required to migrate the AlignSequences example from MBF V1.0 to .NET Bio 1.0 which will help illustrate important changes in .NET Bio.

To update AlignSequences to .NET Bio

1. Step [1] of the example : add the following the **using** statements

using System.Linq;

1. Step [2]: **GenBankParser.ParseOne** method has been removed. Use **GenBankParser.Parse().First().**

Change the following MBF code:

//[2]

ISequence testSequence1 = parser.ParseOne("GenBankSample1.gbk");

ISequence testSequence2 = parser.ParseOne("GenBankSample2.gbk");

To the following .NET Bio code:

//[2]

parser1.Open("GenBankSample1.gbk");

ISequence testSequence1 = parser1.Parse().First();

1. Step [3]: **ISequenceItem** and **ISequence.Statistics.GetCount** have been removed. Use **ISequence** and **SequenceStatistics** instead.

Change the following MBF code:

//[3]

List<ISequenceItem> nucList = dna.LookupAll(true, true, true, true);

Console.WriteLine("Sequence 1\n");

foreach (ISequenceItem item in nucList)

{

Console.WriteLine("{0} = {1}", item.Symbol,

testSequence1.Statistics.GetCount(item.Symbol));

}

To the following .NET Bio code:

//[3]

Console.WriteLine("Sequence 1\n");

SequenceStatistics sequenceStatistics1 = new

SequenceStatistics(testSequence1);

foreach (byte item in dna)

{

Console.WriteLine("{0} = {1}", (char)item,

sequenceStatistics1.GetCount(item));

}

1. Step [4]: **MoleculeType** and **ToString** have been removed. Instead of **testSequence1.ID.ToString** (which is **ISequence.ID.ToString**) use **testSequence1.ID** (which is **ISequence.ID**) and instead of **testSequence1.MoleculeType**.**ToString** use **testSequence1.Alphabet.Name**.

Change the following MBF code:

//[4]

Console.WriteLine("ID = {0}", testSequence1.ID.ToString());

Console.WriteLine("DisplayID = {0}",

testSequence1.DisplayID.ToString());

Console.WriteLine("MoleculeType = {0}",

testSequence1.MoleculeType.ToString());

foreach (Nucleotide nuc in testSequence1)

{

Console.Write(nuc.Symbol);

}

To the following .NET Bio code:

//[4]

Console.WriteLine("ID = {0}", testSequence1.ID);

Console.WriteLine("MoleculeType = {0}",

testSequence1.Alphabet.Name);

foreach (byte nuc in testSequence1)

{

Console.Write((char)nuc);

}

1. Step [5]: To print the nucleotides in each PairwiseAlignedSequences add a *foreach* loop to get the byte. Also remove the **ToString** from each **WriteLine** call.

Console.WriteLine(

"First Sequence: {0}\n", item.FirstSequence.ToString());

1. Step [6]: **FastaFormatter** is changed to **FastAFormatter** and **outputFormatter** has changed, **Format** is no longer used.

Change the following MBF code:

FastaFormatter outputFormatter = new FastaFormatter();

To the following .NET Bio code:

FastAFormatter outputFormatter = new FastAFormatter();

FastAFormatter.Format has been removed so change the following MBF output code:

outputFormatter.Format(outputSequence,"fasta\_out.fasta");

To the following .NET Bio code using a **Write** statement:

outputFormatter.Open("fasta\_out.fasta");

outputFormatter.Write(outputSequence);

outputFormatter.Close();

## AlignSequences Notes

Although AlignSequences is quite simple, it shows how to use some of the key API elements and demonstrates a programming pattern that is used by many Framework applications. The following list—which is keyed to the numbered comments in Listing 1—briefly describes the associated code. The sections following these notes provide a more detailed examination of these key topics.

### [1] Add using Statements for Bio Namespaces

The Bio API has a namespace hierarchy, with Bio as the root namespace and separate child namespaces for the various components.

### [2] Read input data from storage

The Framework includes several parsers, each of which handles a standard data format such as GenBank or FASTA. Each parser reads data and metadata from the associated file type and converts the data to the Framework object model.

AlignSequences uses the **GenBankParser.Parse().First()** to read GenBank-formatted data from two files, each of which contains a single sequence. It converts the data in each file to a Framework **Sequence** objects. It returns **ISequence** interfaces on the objects, which represent the sequences for all subsequent Framework operations.

### [3] Validate Input Data

AlignSequences checks for obvious problems by printing the count of each nucleotide in the sequence. **SequenceStatistics** iterates through the sequence and tracks the number of occurrences of each symbol. **DnaAlphabet** has the list of each nucleotide symbol in the DNA alphabet. AlignSequences uses this list and **SequenceStatistics.GetCount** to print the counts.

### [4] Display information from the input sequences

**ISequence** contains an ordered list of the items in the sequence—nucleotides in this example. AlignSequences print some of the sequence metadata followed by the sequence itself.

### [5] Analyze the input data

After converting the input sequences to Framework objects, you can use Framework algorithms to manipulate or analyze the data in a variety of ways. AlignSequences uses the Needleman-Wunsch alignment algorithm to align the two sequences and produce a consensus sequence.

### [6] Write the results to storage

The Framework includes a set of formatters that write the contents of a **Sequence** or **SequenceRange** object to an appropriately formatted file. .NET Bio Framework is format-independent, so you can write a sequence to any supported format, regardless of the input format. AlignSequences uses the **FastAFormatter** object to write the consensus sequence from Step 5 to a FastA-formatted file.

**Note:** The pattern of creating an object such as **Sequence** to represent data but returning an interface on the object is used throughout the Framework API. For more discussion of this pattern, see “[Object Model: Sequences and Related Types](#_Object_Model:_Sequences)” later in this document.

# .NET Bio Framework Architecture

The following figure illustrates the overall .NET Bio Framework architecture.



.NET Bio Framework architecture

The following is a brief description of each layer. They are described in more detail in subsequent sections.

## Sample Applications and Utilities

.NET Bio Framework includes two applications that use the underlying Framework infrastructure:

* .NET Bio Extension for Excel is an add-in that allows users to work with sequences by using Microsoft Excel.
* .NET Bio Sequence Assembler is a freestanding GUI application that allows users to visualize and manipulate genomic data.

For download links for these applications, see the .NET Bio site on CodePlex, <http://bio.codeplex.com>.

Users can implement their own applications using any .NET-compatible language, including Iron Python.

.NET Bio Framework also supports several utility applications, including the following:

Utilities for .NET Bio

|  |  |
| --- | --- |
| Utility | Description |
| ComparativeUtil | A utility to kick off the comparative assembly. |
| ConsensusUtil | Used for ComparativeUtil step 4. Users can manipulate the data before using it as an input for the next step in the chain. |
| FileFormatConverter | Converts between different file formats. |
| LayoutRefinementUtil | Used for ComparativeUtil step 3. Users can manipulate the data before using it as an input for the next step in the chain. |
| LISUtil | A utility tool for the longest increasing sequence of mummer |
| MumUtil | Optimizations to support large genome assembly. |
| NucmerUtil | Used for ComparativeUtil step 1. Users can manipulate the data before using it as an input for the next step in the chain. |
| PadenaUtil | A utility that defines the scaffolding. |
| ReadSimulator | Produces data in a short-read form, similar to what might be produced by a next-generation sequencing machine |
| RepeatResolutionUtil | Used for ComparativeUtil step 2. Users can manipulate the data before using it as an input for the next step in the chain. |
| SAMUtils | A command-line tool that performs various operations on SAM and BAM-formatted files. |
| ScaffoldUtil | Used for ComparativeUtil step 5. Users can manipulate the data before using it as an input for the next step in the chain. |

## I/O and Analysis

I/O and analysis components both operate on the Framework object model, so they are effectively at the same level in the architecture. However, the two types of component serve very different functions, so they are displayed separately.

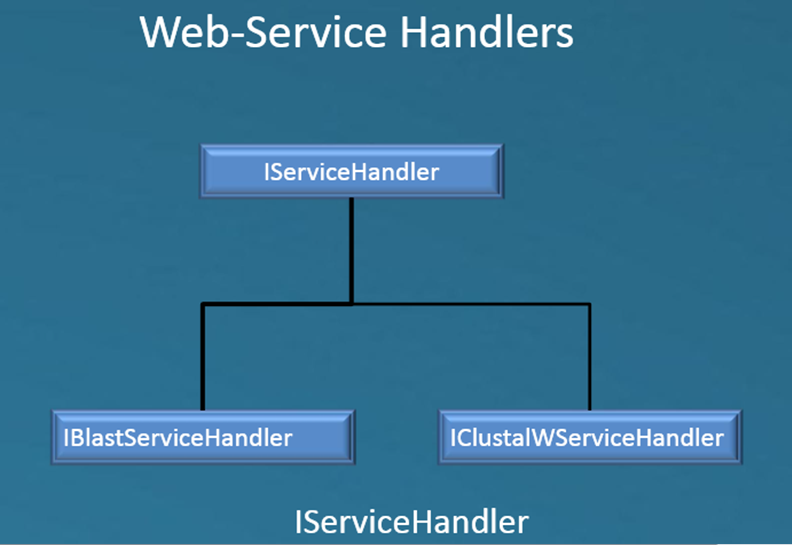
### I/O Components

.NET Bio Framework applications typically start with sequence-related data that is stored in a variety of formats, usually as plain text files. Each format has parser, which reads the input data from storage and converts it to the Framework object model, a format-independent internal representation. Most parsers have a corresponding formatter that converts data from the Framework object model to the associated format and writes the data to storage.

The Framework includes a standard set of parsers and formatters that handle common sequence formats stored as plain text files. Users can extend the Framework by implementing and registering custom parsers and formatters to handle other formats or storage types. For details, see “[Input and Output: Parsers and Formatters](#_Input_and_Output:)” later in this guide.

Web Service connectors transmit Framework sequence data to a remote site for analysis and return the results to the application. Users can extend the Framework by implementing and registering Web Service connectors for other sites and services. The following web services and their service handlers are included in the deployed project:

|  |  |
| --- | --- |
| Web Services | Description |
| BioHPC | Bio\Source\Framework\Bio.WebServiceHandlers |
| EDI | Bio\Source\Framework\Bio.WebServiceHandlers |
| NCBI | Bio\Source\Framework\Bio.WebServiceHandlers |
| BLAST | Handler Bio.Web.Blast.IBlastServiceHandler at Bio\Source\Framework\Bio\Web. |
| ClustalW | Handler Bio.Web.ClustalW.IClustalWServiceHandler at Bio\Source\Framework\Bio\Web. |



### Analysis Components

.NET Bio Framework provides a standard set of components for analyzing sequences in various ways, including

* Sequence alignment, including support for standard algorithms such as Needleman-Wunsch and Smith-Waterman.
* Sequence assembly, including support for standard De Bruin graph techniques in a novel Parallelized De Novo Assembler (Padena).
* Genomic interval techniques for sorting and intersecting two genomic sequence ranges.
* Various utility methods, including logging support.

For more information, see “[Data Processing: Algorithms](#_Data_Processing:_Algorithms)” later in this guide. Users can extend the Framework by implementing and registering custom tools and utilities.

**Caution:** The project library uses zero-based indices consistently across all algorithms, classes, and methods. The purpose of this practice is to make it easier for programmers to work with and extend the library. However, many bioinformatics algorithms and tools use 1-based indices. You must be careful when comparing the output of project tools and functionality with output from similar tools and functionality implemented for other platforms, which might not use 0-based indices.

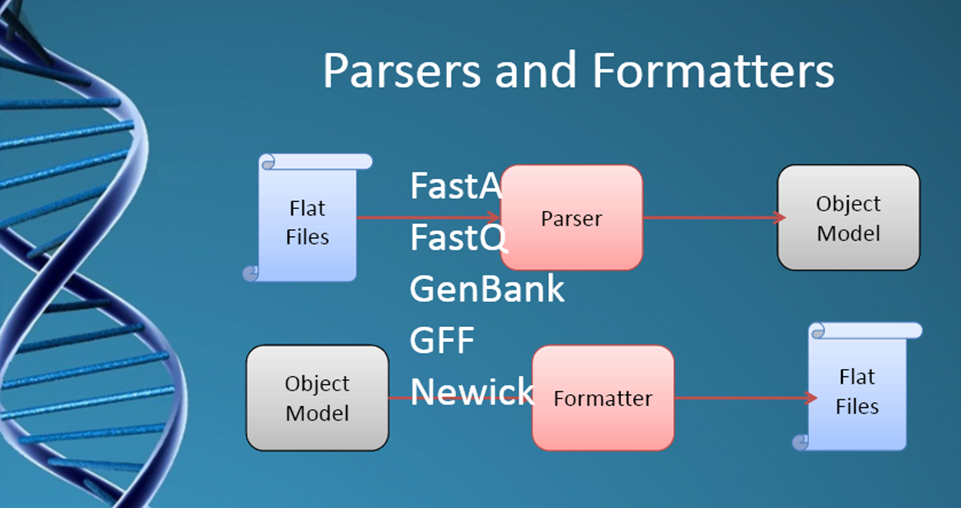
## Object Model

.NET Bio Framework uses a format-independent object model to handle sequence data. The model includes objects to represent:

* A variety of different sequences, including DNA, RNA, and proteins.
* Genomic intervals.
* Alphabets, including DNA, RNA, or protein alphabets.
* Encoding, to store sequence data in a variety of compressed formats.
* Phylogenetic trees.
* Matrix data, such as BLOSUM45.

# Input and Output: Parsers and Formatters

Sequence-related data is typically stored as plain text files in a variety of formats. The project parsers and formatters handle the task of reading data from and writing it to storage, respectively. Although they are at opposite ends of the architecture, they perform closely-related tasks, so they are both discussed in this section.



The first step for most Framework applications is to use a parser to read the data from storage and convert it to the Framework object model, such as **Sequence** or **SequenceRange** objects. Those objects can then be used by subsequent Framework operations.

Most parsers have a corresponding formatter that writes the data from the object model to storage in the appropriate format. Because Framework stores sequence data in a format-independent way, you can write the data to storage in any appropriate format, regardless of the input format. In fact, one simple way to use the project is to implement a format converter.

The following table describes the standard parsers and formatters supported by the Framework. Each handles a single format for data stored in plain text files. The format name is linked to a Web site that describes the format. The parser and formatter for most of the supported formats are in separate namespaces, named for the format. For example, the GenBank parser and formatter are in the **Bio.IO.GenBank** namespace. The exception is **SnpParser**, which is in the **Bio.IO** namespace. Types that support all parsers and formatters are in the **Bio.IO** namespace.

Parsers and Formatters

|  |  |  |  |
| --- | --- | --- | --- |
| Format | Data Type | Formatter | Returns |
| [BED](http://genome.ucsc.edu/FAQ/FAQformat) | Genomic intervals | Yes | **IList<ISequenceRange>** |
| [ClustalW](http://www.ebi.ac.uk/Tools/clustalw2/index.html) | Sequence alignment | No | **IList<ISequenceAlignment>** |
| [FASTA](http://www.pnas.org/content/85/8/2444.long) | Sequence | Yes | **IEnumerable<ISequence>** |
| [FASTQ](http://maq.sourceforge.net/fastq.shtml) | Sequence | Yes | **IEnumerable<IQualitativeSequence>** |
| [GenBank](http://www.ncbi.nlm.nih.gov/books/bookres.fcgi/handbook/ch1.pdf) | Sequence | Yes | **IEnumerable<ISequence>** |
| [GFF](http://www.sanger.ac.uk/Software/formats/GFF/GFF_Spec.shtml) | Sequence | Yes | **IEnumerable<ISequence>** |
| [Newick](http://evolution.genetics.washington.edu/phylip/newicktree.html) | Phylogenetic | Yes | **Tree** |
| [Nexus](http://en.wikipedia.org/wiki/Nexus_file) | Sequence alignment | No | **IList<ISequenceAlignment>** |
| [Phylip](http://en.wikipedia.org/wiki/PHYLIP) | Sequence alignment | No | **IList<ISequenceAlignment>** |
| [SAM](http://samtools.sourceforge.net/SAM1.pdf) | Sequence alignment | Yes | **IList<ISequenceAlignment>** |
| SNP | SNP items | No | **IEnumerable<ISequence>** |
| Simplesnp |  |  |  |
| [BAM](http://samtools.sourceforge.net/SAM1.pdf) | Sequence alignment | Yes | **IList<ISequenceAlignment>** |
| XsvTextReader |  |  | XSV related Parser and formatters |
| XsvSparseReader |  |  | XSV related Parser and formatters |
| XsvSparseParser |  |  | XSV related Parser and formatters |
| XsvSparseFormatter |  | Yes | XSV related Parser and formatters |
| XsvSnpReader |  |  | XSV related Parser and formatters |
| XsvContigParser |  |  | XSV related Parser and formatters |
| XsvContigFormatter |  | Yes | XSV related Parser and formatters |

**Notes:** The Returns column lists the interface returned by the parser’s **Parse().First()** method. There are two exceptions:

* The BED parser exposes **ParseRange** and **ParseRangeGrouping** methods rather than **Parse** and parser’s **Parse().First()**. The table lists the return value of **BedParser.ParseRange**.
* The Newick parser exposes only a **Parse** method, and returns a **Bio.Phylogenetics**.Tree object rather than an interface.

Users can extend the Framework by implementing custom parsers and formatters to handle data in other formats or storage types, and registering them with the Framework.

## Parsers

Parser names typically use the format name followed by **Parser**, such as **GenBankParser**.

### How to use a deployed parser

To use a .NET Bio Framework parser

1. Create a parser object for the input format.

GenBankParser parser1 = new GenBankParser();

2. Pass the file to the appropriate parsing method, along with any related information.

parser1.Open("GenBankSample1.gbk");

ISequence testSequence1 = parser1.Parse().First();

The parser reads the file and packages the contents in the appropriate objects. For example, sequence parsers such as **FastaParser** create one or more **Sequence** objects—one for each sequence in the file—and return an **ISequence** interface for each **Sequence** object.

**Tip:** For sequence data, if you do not know the file format, try passing the file name to the **Bio.IO.SequenceParsers.** **FindParserByFileName()** method. The method will attempt to determine the format and, if successful, return an **ISequenceParser** interface to the appropriate parser. However, this approach can reduce performance, so it should be used only if necessary.

**FindParserByFileName()** finds the parser and opens the file.

The **FindParserByFileName** constructorcalls the appropriate parser, for example, **FastAParser**() with the file name:

if (IsFasta(fileName))

{

parser = new FastAParser(fileName);

}

**FastAParser**() in turns calls **Open**() with the file name:

public FastAParser(string filename)

{

this.Open(filename);

}

**Note:** Since the **FindParserByFileName** constructor opens the specified file, if you call **Open()** on the same filename an exception is thrown. Well-designed code will not call **Open()** on an already open file.

The details depend on the input files, the particular parser, and the associated format.

* Most parsers support two standard input methods, **Parse().First()** and **Parse**, which handle files that contain single and multiple sequences. Parse returns **IEnumerable<ISequence>**.

**Parse().First()** returns a single interface and **Parse** returns **IEnumerable<ISequence>**. If you aren’t sure how many sequences are in the file, use **Parse**, which can also handle files that contain a single sequence.

* The input methods have several overloads to accommodate different input types, including: path strings and **TextReader** objects.
* The input methods allow you to specify whether the returned interface is read-only.

### Custom Parsers

For non-standard file formats and storages, you must implement a custom parser that retrieves the data from storage, packages the data as Framework objects, and returns the appropriate Framework interfaces. The details are beyond the scope of this document, but if you download the Framework sources, you can use the standard parsers as a guide. Each parser has its own Visual Studio project, and the parser projects are all under the source tree’s **Bio\IO** folder.

To use your custom parser simply instantiate the classes directly:

var parser = new MyCustomParser();

Alternatively, developers can expose their classes as add-ins which allows alphabets, parsers, formatters and algorithms to be added dynamically to .NET Bio Framework collections:

var parser = SequenceParsers.All.FirstOrDefault(sp => sp.FileTypes.Contains(Path.GetExtension(filename)));

Using the add-in option enables you to lookup the parser using the file extension. You do not have to know exactly which one it should be as long as it exposes **FileTypes** properly. The primary reason why you want to expose your classes as add-ins is they become dynamically discoverable – i.e. existing programs such as the Excel add-in and sequence assembler can access your parsers without being recompiled or changed to support them. It provides reuse without change and hides the implementation.

## Formatters

Most parsers have a corresponding formatter, which writes the contents of one or more Framework objects to an appropriately formatted file. Formatter names typically use the format name followed by **Formatter**, such as **GenBankFormatter**. Formatters usually have a single method, **Format**, with several overloads that allow you to:

* Use a file path or a **TextWriter** object to handle the output.
* Format a single sequence or a collection of sequences.

If you implement a custom parser, you usually also implement a corresponding custom formatter. For examples, see the parser projects in the source tree.

How to use an existing formatter

1. Create a sequence with content.

string seq = @"GACGCCGCCGCCACCACCGCCACCGCCGCAGCAGAAGCAGCGCACCGCAGGAGGGAAG" +

"ATGCCGGCGGGGCACGGGCTGCGGGCGCGGACGGCGACCTCTTCGCGCGGCCGTTCCGCAAGAAGGGTTA" +

"CATCCCGCTCACCACCTACCTGAGGACGTACAAGATCGGCGATTACGTAGACGTCAAGGTGAACGGTG";

Sequence outputSequence = new Sequence(Alphabets.DNA, seq);

1. Instantiate a FastA formatter and open a file to write to.

FastAFormatter outputFormatter = new FastAFormatter();

outputFormatter.Open("fasta\_out.fasta");

1. Write the contents of the **outputSequence** to the "fasta\_out.fasta" file and close the file.

outputFormatter.Write(outputSequence);

outputFormatter.Close();

# Input and Output: Web Service Connectors

One way to perform certain types of sequence analysis is to submit data to a remote site, which processes the data and returns the results. For example, you can submit sequences to a Basic Alignment Search Tool (BLAST) Web site, which looks for regions of local similarity between its database of sequences and your sample.

Some Web sites, including several that support BLAST requests, allow you to use Web services to submit requests. .NET Bio Framework includes a Web service API that simplifies the process of submitting a sequence for analysis. The following is a general procedure for using the Framework Web service API to submit sequences to remote sites.

To submit data to a remote site

1. Create a Framework object that contains the sequence to be submitted.

2. Create and configure a Web Services connector object—sometimes called a service handler—for the Web site.

Each supported Web site has a separate connector.

3. Use the connector object to submit the request.

4. Retrieve the results.

For a detailed example based on a simple console application, see “[Example: How to Submit a Framework Web Services Request](#_Example:_How_to)” later in this paper.

**Tip:** Web service requests sometimes fail and, when they do succeed, can take a significant amount of time to process the data and return results. Your application should be able to handle both scenarios.

Currently, .NET Bio Framework includes connectors for the following BLAST Web services. You can extend the Framework by implementing and registering connectors for other Web Services.

Web Service Connectors

|  |  |
| --- | --- |
| Type Name | Web Site |
| EbiWuBlastHandler | [http://www.ebi.ac.uk/WSWUBlast#runWUBlast](http://www.ebi.ac.uk/WSWUBlast%23runWUBlast) |
| NCBIBlastHandler | <http://www.ncbi.nlm.nih.gov/blast/Blast.cgi> |

**Note:** The Framework includes code to support the AzureBlast service. However, the service is used only for evaluation and demonstration purposes, and is not usually available.

# Object Model: Sequences and Related Types

A basic change has been made to the Object Model. It now uses **ISequence:IEnumerable<byte>**. Implementations of **ISequence** make up one of the core sets of data structures in Bio that store data relevant to DNA, RNA, and Amino Acid structures. Several algorithms for alignment, assembly, and analysis take these items as their basic data inputs and outputs.

The **Sequence** class is the standard implementation of the **ISequence** interface. It contains the raw data that defines the contents of a sequence. Since **Sequence** uses enumerable of bytes that can be accessed as follows:

Sequence mySequence = new Sequence(Alphabets.DNA, "GATTC");

foreach (byte nuc in mySequence) { ... }

The results will be based on the Alphabet associated with the sequence. Common alphabets include those for DNA, RNA, and Amino Acids. **Sequence** also provides a means to for users to access the underlying data directly. This may be useful for those writing algorithms against the sequence where performance is especially important.

A sequence object is basically built from the **IAlphabet** and **ISequence** interfaces. They are the core of the Framework API and we recommend coding against the interfaces versus types. The **ISequence** interface is essentially an efficient format-independent way to store sequence data and metadata and represent that data in the Framework environment. Each sequence has an associated alphabet that defines the available symbols.

The following figure illustrates the sequence object model using **ISequence** to describe a sequence of symbols and **IAlphabet** to describe the alphabet used to define the nuleotide and protein:



## Alphabets

An alphabet defines the available set of symbols for a sequence element. For example, the full DNA alphabet support symbols for:

* The four nucleotides—adenine, cytosine, guanine and thymine—which are usually represented by one-character abbreviations—A, C, G, and T.
* Degenerate base symbols, which represent ambiguous elements that could contain any of two or more nucleotides.

For example, M indicates that the element could be either A or C.

* Gaps in the sequence, which are usually indicated by a “-” (hyphen) character.

.NET Bio Framework supports three alphabets:

Alphabet Represented by

DNA **Bio.DnaAlphabet**

RNA **Bio.RnaAlphabet**

Protein **Bio.ProteinAlphabet**

For simplicity, this document focuses primarily on the DNA alphabet. The RNA and Protein alphabets are listed in [Appendix B](#_Appendix_B:_RNA).

**DnaAlphabet** contains a set of fields to represent every symbol as a byte.

## The Sequence Object

From an application perspective, a **Sequence** object is basically a container for a sequence and its metadata. A **Sequence** object is usually represented by an **ISequence** interface. For example, sequence parsers return **ISequence** interfaces, not the objects themselves.

**Note:** Parsers are required only to return an object that supports **ISequence**. As a practical matter, parsers typically return **Sequence** objects.

**ISequence** supports only a subset of the object’s properties and methods.

**Note**: **ISequence** (**System.Collections.Generic.IEnumerable<out T>**) is read-only.

Each sequence element is represented by a **byte** object from **DnaAlphabet**. **Sequence** is an indexed object, which allows you to enumerate the nucleotides in a DNA sequence as shown in the following example snippet.

foreach (byte item in mySequence)

{

Console.WriteLine("{0} = {1}", (char)item,

seqStat.GetCount(item));

}

### How to Create a Sequence Object

There are two basic ways to create a **Sequence** object. The most common approach, as discussed earlier, is to use a parser to read the data from a file, create the objects, and return the objects’ **ISequence** interfaces.

To obtain a sequence by using a parser

1. Applications typically obtain **Sequence** objects—or more accurately, their **ISequence** interfaces—from parsers as shown in the following snippet from the AlignSequences example.

GenBankParser parser1 = new GenBankParser();

parser1.Open("GenBankSample1.gbk");

ISequence testSequence1 = parser1.Parse().First();

However, you can also create **Sequence** objects from scratch. This is the standard implementation of the **ISequence** interface.

**Sequence** has several constructors, which allow you to specify:

* A string containing the sequence data.
* Optionally, the alphabet that is associated with the sequence.

If you don’t specify an alphabet, **Sequence** uses **DnaAlphabet** by default.

To create a sequence object from scratch

1. Create a Sequence object for a simple DNA sequence, GATTCCA. For simplicity, the example uses a string literal and explicitly specifies the DNA Alphabet.

Sequence mySequence = new Sequence(Alphabets.DNA, "GATTCCA");

### How to Enumerate a Sequence

**Sequence** contains the raw data that defines the contents of a sequence using enumerable of bytes. The simplest way to enumerate a sequence is with a C# **foreach** loop.

To enumerate a sequence

1. Create a Sequence object for a simple DNA sequence, GATTCCA. For simplicity, the example uses a string literal and explicitly specifies the DNA Alphabet.

Sequence mySequence = new Sequence(Alphabets.DNA, "GATTCCA");

1. Generate sequence statistics, the number of occurrences of each symbol within the sequence.

SequenceStatistics seqStat = new SequenceStatistics(mySequence);

1. Now iterate through each byte (symbol) of the **mySequence** sequence and write the count generated by **SequenceStatistics** for each symbol.

foreach (byte item in mySequence)

{

Console.WriteLine("{0} = {1}", (char)item,

seqStat.GetCount(item));

}

The results will be based on the Alphabet associated with the sequence. Common alphabets include those for DNA, RNA, and Amino Acids.

### Sequence Metadata

Sequences are usually accompanied by a variety of metadata that provide context for the sequence, such as the source of the data, the authors of the associated study, and so on.

Sequence exposes some standard metadata as properties, as listed in in the following table.

Sequence Metadata Properties

|  |  |
| --- | --- |
| Property | Description |
| **Alphabet** | The alphabet to which symbols in this sequence belong. |
| **Count** | The number of bytes contained in the Sequence |
| **ID** | An identifier. ID is usually just a brief code to distinguish this sequence from others. |

In addition to the standard metadata represented by the **Sequence** properties, data files typically contain an unpredictable variety of nonstandard metadata that varies from format to format. The Framework stores nonstandard metadata in a generic **Dictionary** object.

A **Dictionary** object is basically a container for a collection of key-value pairs.

* The key is a string that identifies the data.
* The value is an **Object** type that can contain any object.

The dictionary is stored in the **Sequence.Metadata** property. To retrieve an item, specify the associated key, and cast the result to the appropriate type. The following example retrieves the Authors metadata from the *mySequence* object.

Dictionary<string, object> myMetadata = mySequence.Metadata;

string authors = (string) myMetadata[“Authors”];

Each parser is responsible for storing metadata in the dictionary, so you should consult the parser documentation or source code for details.

### Specialized Sequence objects

So far, this document has focused on the **Sequence** object, which is the most commonly used object for representing sequences. There are also several related objects—all of which also expose **ISequence**—that are used for specialized purposes:

ISequence objects

|  |  |
| --- | --- |
| Derived object | Description |
| **DerivedSequence** | Represents a sequence that is derived from another sequence. It allows you to access the complement or reversal of the source sequence without storing the data in memory twice. |
| **QualitativeSequence** | Represents sequence data with quality scores. It is the basis for the FASTQ format. |
| **Sequence** | The most commonly used object for representing sequences. |
| **SparseSequence** | Represents discontinuous sequences. It is typically used when a sequence is quite large, but contains only a small amount of interesting data. **SparseSequence** stores its data by index, which provides better performance than storing the sequence as a list or array. |

For more information on these objects, see the .NET Bio Help file, Bio.chm.

## Sequence Manipulation

There are two types of user who may wish to manipulate sequences.

* Scientists:  
  Sequences are read only  
  You cannot insert or remove elements within a sequence.  
  You can build a new sequence.  
  The object model provides the following methods for operations on a sequence:

Sequence.GetComplementedSequence

Sequence.GetReverseComplementedSequence

Sequence.GetReverseSequence

Sequence.GetSubSequence

* Developers  
  A developer can programmatically use C# byte array manipulation to manipulate a sequence.  
  To a manipulate a byte array using C:#

/// Converting a string to byte array.

System.Text.ASCIIEncoding encoding = new

System.Text.ASCIIEncoding();

byte[] myBytes = encoding.GetBytes(myString);

/// Converting a byte array back to a string:

System.Text.ASCIIEncoding encoding = new

System.Text.ASCIIEncoding();

string myString = encoding.GetString(myBytes);

/// Getting the elements 5 through 10 in an byte array

System.Text.ASCIIEncoding encoding = new

System.Text.ASCIIEncoding();

byte[] myBytes = encoding.GetBytes("01234fetch999");

byte[] mySubArray = new byte[5];

System.Array.Copy(myBytes, 5, mySubArray, 0, 5);

The **ISequence** interface includes methods and properties that you can use to manipulate sequences in various ways.

**Note**: All Sequences are meant to be read only because they are a byte array. You cannot add, remove or change an existing sequence.

The following table lists the relevant methods.

Sequence Manipulation Methods

|  |  |
| --- | --- |
| Method | Description |
| **Byte** | Returns the byte found at the specified index if within bounds. |
| **GetReversedSequence** | Return a new sequence representing this sequence with the orientation reversed. |
| **GetComplementedSequence** | Return a new sequence representing the complement of this sequence. |
| **GetReverseComplementedSequence** | Return a new sequence representing the reverse complement of this sequence. |
| **GetSubSequence** | Return a new sequence representing a range (subsequence) of this sequence. |
| **IndexOfNonGap(0)** | Gets the index of first non-gap symbol. |
| **IndexOfNonGap(long startpos)** | Returns the position of the first symbol beyond **startPos** that does not have a Gap symbol. |
| **LastIndexOfNonGap()** | Gets the index of last non-gap symbol |
| **LastIndexOfNonGap(long endPos)** | Returns the index of last non-gap symbol before the specified end position. |
| **GetEnumerator()** | Gets an enumerator to the bytes present in this sequence |

For an example of how to use these methods, see “[Example: How to Manipulate a Sequence](#_Example:_How_to_1)” later in this document.

## The SequenceRange Object

Most of the formats that .NET Bio Framework supports describe a complete sequence. However, it is also useful to represent genomic interval data rather than explicit sequences. The Framework uses **SequenceRange** objects or the **ISequenceRange** interface to represent genomic intervals. In particular, the Framework represents the data from BED formatted files by **ISequenceRange** interfaces.

A **SequenceRange** object contains the data required to represent a region within a parent sequence. The region is defined by a start and end index, relative to the original sequence.

## The AlignedSequence Object

An **AlignedSequence** object and the associated **IAlignedSequence** interface represent aligned sequences. These objects are represented through the **ISequenceAlignment** interface (Bio.Algorithms.Alignment.ISequenceAlignment), which is basically a container for a list of **IAlignedSequence** interfaces. This is just a storage object. It is up to an algorithm object to fill it in.

**IAlignedSequence** supports two properties.

|  |  |
| --- | --- |
| Property | Description |
| **Metadata** | Information such as the alignment score, offsets, consensus, and so on. |
| **Sequences** | A list of the aligned sequences. |

Several of the Framework parsers, such as those for the ClustalW and Nexus formats, return the data as **ISequenceAlignment** or related interfaces.

# Object Model: Other Types

The Framework object model includes several types that are used for specialized purposes, including phylogenetic trees, single-nucleotide polymorphism (SNP) items, and matrix data. This section briefly describes these types.

## Phylogenetics

A phylogenetic tree describes evolutionary relationships between organisms that derive from a common ancestor. Each organism is represented as a node in the tree. The nodes are connected by “edges”, the length of which sometimes represents time estimates. The following figure shows a schematic version of a simple phylogenetic tree.



Phylogenetic tree

The Framework object model represents trees as follows:

* A tree is represented as a **Bio.Phylogenetics.Tree** object, which contains the tree’s root node.
* A node is represented as a **Bio.Phylogenetics.Node** object. Node objects expose properties that provide links to the node’s child nodes and associated edges.
* An edge is represented as a **Bio.Phylogenetics.Edge** object, which has a **Distance** property that contains the edge length.

The Newick parser, **Bio.IO.Newick.NewickParser**, reads phylogenetic trees stored as Newick-formatted files and returns a **Bio.Phylogenetics.Tree** object that represents the tree. You can then start with the root node and “walk” the tree to obtain the complete tree.

## SNP Items

SNP items, SNPs (single nucleotide polymorphism), represent sequence variations between species or paired chromosomes. The Framework represents SNP items as **SparseSequence** objects. For more details on SNP, see “Single-nucleotide polymorphism.”

The SNP parser, **SnpParser**, reads SNP data from a file and returns a **SparseSequence** object for each SNP item in the file.

## Bio.Matrices

Bioinformatics uses matrices in a variety of ways. The **Bio.Matrix** namespace provides general-purpose support for matrix-related techniques. The following figure shows a schematic representation of a Bio matrix.



Bio matrix

The unshaded part represents the matrix proper and the shaded part represents the associated row and column keys. This representation allows you to access the data in either of two ways:

* By conventional row and column indices.

For example:

**Nucleotide** data = bioData[1, 2];.

* By keys, which are strings that describe the contents of the column or row.

For example:

**Nucleotide** data = bioData[“keyR1”, “keyC2”];

For more details, see the **Bio.Matrix** namespace in the .NET Bio Help File, Bio.chm.

# Data Processing: Algorithms

After you have created one or more sequence objects, you can manipulate or analyze the sequence data in various ways. The algorithm output is typically algorithm specific and often generates either another sequence or statistics and comparison results.

.NET Bio Framework provides support for the following standard bioinformatics algorithms:

* sequence alignment
* sequence assembly
* translation
* sequence searching

The Framework also supports add-in algorithms for 3rd party extensions and enables you to make them visible through add-in registration. Almost everything is defined in terms of interfaces which allows greater transparency and allows for extension and versioning.

* The **Sequence** object supports methods and properties that you can use to manipulate sequences.
* **Bio.Algorithms** and its child namespaces contain a collection of types that you can use to analyze sequences in various ways, and produce derived data such as k-mers from sequence data.

The following list briefly describes the contents of **Bio.Algorithms** and its child namespaces. For details, see the .NET Bio Help file or the source code. For an example of how to manipulate sequences, see the next section.

**Note:** The methods that run the various algorithms such as **NeedlemanWunschAligner.Align** are all synchronous and cannot be canceled. After you call the method, the algorithm runs until it is finished unless you cancel the process or thread. Consider running time-consuming algorithms in a separate thread. This prevents the algorithm from blocking the primary thread and allows you to terminate the algorithm by canceling the thread.

#### Bio.Algorithms

This namespace includes types that support algorithms, including the **SequenceToKmerBuilder** type, which constructs k-mers from sequences, and the **KurtzSuffixTreeBuilder** type, which builds a suffix tree by using the Kurtz algorithm.

#### Bio.Algorithms.Alignment Namespace

This namespace includes a collection of types that support a variety of standard sequence alignment algorithms, as summarized in the following table.

Sequence Alignment Types

|  |  |
| --- | --- |
| Property | Description |
| **LongestIncreasingSubsequence** | Finds the longest increasing subsequence from a list of maximum unique matches (MUMs). |
| **MUMmerAligner** | A system for rapidly aligning entire genomes or entire sequences. |
| **NeedlemanWunschAligner** | Implements the Needleman-Wunsch algorithm for global alignment. |
| **NUCmerPairwiseAligner** | A system for rapidly aligning entire genomes or very large DNA sequences. |
| **SmithWatermanAligner** | Implements the Smith-Waterman algorithm for local alignment. |

#### Bio.Algorithms.Assembly

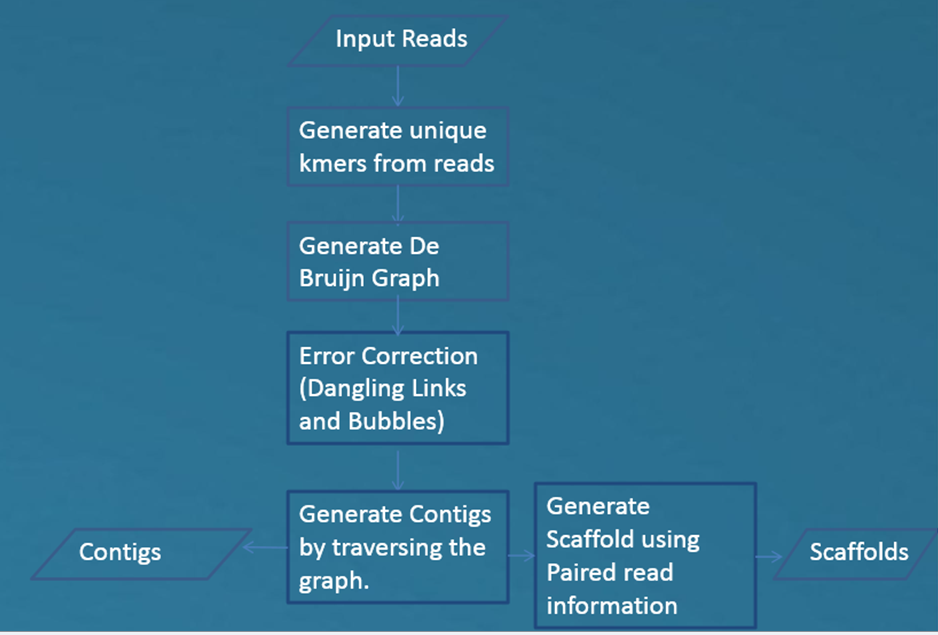
This namespace contains types that support assemblies, including the **OverlapDeNovoAssembler** type, which Implements a simple greedy DNA assembly algorithm.

#### Bio.Algorithms.Assembly.Graph

This namespace supports de Bruijn graphs.

#### Bio.Algorithms.Assembly.Padena (Parallel DeNovo Assembler)

This namespace supports a De Novo assembly algorithm, which is based on the techniques published for ABYSS, VELVET, and EULER-SR. For maximum efficiency on multi-core desktop computers, the algorithm is parallelized by using the .NET Framework 4.0 parallel extensions.



#### Bio.Algorithms.Alignment.MultipleSequenceAlignment

The namespace includes the **PAMSAMMultipleSequenceAligner** type, which implements a parallelized version of multiple sequence alignment algorithm based on the techniques published by MUSCLE. For further information, see the PAMSAM sample.

For maximum efficiency on multi-core desktop computers, the algorithm is parallelized by using the .NET Framework 4.0 parallel extensions.

#### Bio.Algorithms.Translation

This namespace contains several types that support translation, as listed in the following table.

Sequence Alignment Types

|  |  |
| --- | --- |
| Type | Description |
| **Codons** | Contains a table of mappings from RNA nucleotide triplets to amino acids. |
| **ProteinTranslation** | Supports translating RNA sequences into amino acid sequences. |
| **Transcription** | Supports basic nucleotide transcription across DNA and RNA sequences. |

# Example: How to Manipulate a Sequence

This section demonstrates some of the basics of how to use .NET Bio Framework to manipulate a sequence, including:

* Extract a segment from a larger sequence.
* Generate reverse, complement, and reverse complement sequences.

**Note**: This represents a change from earlier versions. In version 2.0 you cannot modify symbols, add or delete symbols. You would have to you create another sequence with the deleted or modified symbols.

SequenceManipulation is a simple console application that shows how to perform these basic manipulation tasks. Listing 2 gives the complete source code. The numbered comments identify the key parts of the code and are discussed in the notes that follow the example. For directions on how to build and run the application, see “[A .NET Bio Framework Quick Start](#_A_Bio_Framework)” earlier in this guide.

Listing 2: SequenceManipulation

using System;

using Bio;

using System.Text;

namespace SequenceManipulation

{

class SequenceManipulation

{

static void Main(string[] args)

{

int segmentStart = 0;

int segmentLength = 25;

ISequence segment;

// [1]

string seq = @"GACGCCGCCGCCACCACCGCCACCGCCGCAGCAGAAGCAGCGCACCGCAGGAGGGAAG" +

"ATGCCGGCGGGGCACGGGCTGCGGGCGCGGACGGCGACCTCTTCGCGCGGCCGTTCCGCAAGAAGGGTTA" +

"CATCCCGCTCACCACCTACCTGAGGACGTACAAGATCGGCGATTACGTAGACGTCAAGGTGAACGGTG";

Sequence sequence = new Sequence(Alphabets.DNA, seq);

// [2]

segment = sequence.GetSubSequence(segmentStart, segmentLength);

Console.WriteLine("Segment: {0}", GetString(segment));

// [3]

Console.WriteLine("Reverse Segment: {0}",

GetString(segment.GetReversedSequence()));

Console.WriteLine("Segment Complement: {0}",

GetString(segment.GetComplementedSequence()));

Console.WriteLine("Segment Reverse Complement: {0}\n",

GetString(segment.GetReverseComplementedSequence()));

Console.ReadKey();

}

/// <summary>

/// Gets the string representing this sequence.

/// </summary>

/// <param name="sequence">Sequence instance.</param>

/// <returns>Returns string representing the sequence.</returns>

private static string GetString(ISequence sequence)

{

char[] symbols = new char[sequence.Count];

for (long index = 0; index < sequence.Count; index++)

{

symbols[index] = (char)sequence[index];

}

}

return new string(symbols);

}

}

}

The output is:

Segment: GACGCCGCCGCCACCACCGCCACCG

Reverse Segment: GCCACCGCCACCACCGCCGCCGCAG

Segment Complement: CTGCGGCGGCGGTGGTGGCGGTGGC

Segment Reverse Complement: CGGTGGCGGTGGTGGCGGCGGCGTC

## Migrating the SequenceManipulation example from MBF to .NET Bio

The following highlights the code revisions required for migrating from MBF to .NET Bio.

Listing 2: MBF v1 SequenceManipulation

using System;

using MBF;

namespace SequenceManipulation

{

class SequenceManipulation

{

static void Main(string[] args)

{

int segmentStart = 0;

int segmentLength = 25;

ISequence segment;

// [1]

string seq = @"GACGCCGCCGCCACCACCGCCACCGCCGCAGCAGAAGCAGCGCACCGCAGGAGGGAAG" +

"ATGCCGGCGGGGCACGGGCTGCGGGCGCGGACGGCGACCTCTTCGCGCGGCCGTTCCGCAAGAAGGGTTA" +

"CATCCCGCTCACCACCTACCTGAGGACGTACAAGATCGGCGATTACGTNGACGTCAAGGTGAACGGTG";

Sequence sequence = new Sequence(Alphabets.DNA, seq);

//sequence.IsReadOnly = false;

// [2]

segment = sequence.Range(segmentStart, segmentLength);

Console.WriteLine("Segment: {0}", segment.ToString());

// [3]

Console.WriteLine("Reverse Segment: {0}", segment.Reverse.ToString());

Console.WriteLine("Segment Complement: {0}", segment.Complement.ToString());

Console.WriteLine("Segment Reverse Complement: {0}\n",

segment.ReverseComplement.ToString());

// [4]

sequence[0] = Alphabets.DNA.A;

Console.WriteLine("Modified segment: {0}", segment.ToString());

sequence.Insert(4, Alphabets.DNA.T);

Console.WriteLine("Augmented segment 1: {0}", segment.ToString());

sequence.InsertRange(4, "AAA");

Console.WriteLine("Augmented segment 2: {0}", segment.ToString());

sequence.RemoveRange(4, 4);

sequence[0] = Alphabets.DNA.G;

Console.WriteLine("Original segment: {0}", segment.ToString());

Console.ReadKey();

}

}

}

The original output was:

Segment: GACGCCGCCGCCACCACCGCCACCG

Reverse Segment: GCCACCGCCACCACCGCCGCCGCAG

Segment Complement: CTGCGGCGGCGGTGGTGGCGGTGGC

Segment Reverse Complement: CGGTGGCGGTGGTGGCGGCGGCGTC

Modified segment: AACGCCGCCGCCACCACCGCCACCG

Augmented segment 1: AACGTCCGCCGCCACCACCGCCACC

Augmented segment 2: AACGAAATCCGCCGCCACCACCGCC

Original segment: GACGCCGCCGCCACCACCGCCACCG

The following changes are made for .NET Bio:

SequenceManipulation MBF to .NET Bio changes

|  |  |  |
| --- | --- | --- |
| V1 | V2 | Description |
| **Reverse** | **GetReversedSequence()** | Reverse a sequence. |
| **Complement** | **GetComplementedSequence()** | Complement a sequence. |
| **ReverseComplement** | **GetReverseComplementedSequence()** | Reverse and complement a sequence. |
| **Range** | **GetSubSequence()** | Gets a segment of a sequence with specified start and length. |
| **Sequence.Insert** | **Obsolete** | API removed. Cannot edit sequences. |
| **Sequence.InsertRange** | **Obsolete** | API removed. |
| **Sequence.RemoveRange** | **Obsolete** | API removed. |

## SequenceManipulation Notes

The following list—which is keyed to the numbered comments in Listing 2—briefly describes the associated code.

### [1] Create a Sequence Object

For simplicity, this example creates a sequence by using the **seq** string literal.

string seq =

@"GACGCCGCCGCCACCACCGCCACCGCCGCAGCAGAAGCAGCGCACCGCAGGAGGGAAG" +

Sequence sequence = new Sequence(Alphabets.DNA, seq);

Typically, you would use a parser to read the sequence from a data file. Create a sequence by using a parser to read the sequence from a data file:

ISequence sequence = parser.ParseOne("GenBankSample1.gbk");

**Sequence** objects are read-only, once created you cannot add more nucleotides to a sequence.

For more details about using **ISequence** interface, see “[Parsers](#_Parsers)” earlier in this document.

### [2] Extract a Segment from the Sequence

It is often more convenient to work with a segment of a longer sequence. To create a segment of any length use **Sequence.GetSubSequence**, and provide the segment’s starting index and length. Step 1 in the SequenceManipulation example creates **Sequence** containing the first 25 symbols of the **seg** sequence. Step 2 creates a sequence segment.

segment = sequence.GetSubSequence(segmentStart, segmentLength);

**GetSubSequence** returns a new **Sequence**. This new sequence basically represents a read-only view of the underlying sequence. You can perform operations, such as writing or generating a complement, but you cannot modify the segment directly. You must instead read a new underlying sequence. However, when you do, the contents of the segment change to reflect the modified sequence.

### [3] Generate a Reverse, Complement, and Reverse Complement

**Sequence** and **ISequence** support methods that contain the reverse, complement, and reverse complement sequences. To limit memory requirements, the method actually generate the derived sequences on demand rather than storing the derived sequences.

These methods return new objects and do not modify the original sequence, so you can use them for segments as well as complete sequences. SequenceManipulation generates reverse, complement, and reverse complement sequences for the segment created in Step 2, and displays the results.

Console.WriteLine("Reverse Segment: {0}",

GetString(segment.GetReversedSequence()));

Console.WriteLine("Segment Complement: {0}",

GetString(segment.GetComplementedSequence()));

Console.WriteLine("Segment Reverse Complement: {0}\n",

GetString(segment.GetReverseComplementedSequence()));

SequenceManipulation uses a private method **GetString** to get the string representing this sequence in order to write it to the console.

# Example: How to Submit a .NET Bio Framework Web Services Request

This section introduces the basic features of the Framework Web service API and programming model by walking you through a simple console application, BlastRequest, which submits a sequence to the EBI BLAST service. For more information on the service, see the “EBI BLAST Service” in “Resources.”

BlastRequest uses the following programming pattern, which is used by many Framework Web Service applications:

1. Define the sequence to be tested.

2. Create and configure a service handler.

3. Define the request.

4. Submit the request.

5. Process the returned data.

If you have installed MBF, you can build and run BlastRequest as follows.

To build and run BlastRequest

1. Open Visual Studio 2010 and create a new .NET console application.

2. Open program.cs and replace the contents with the code from Listing 2 in the following section, “BlastRequest Sample.”

3. Add references to Bio.dll and Bio.WebServiceHandlers.dll.

4. If necessary, open the project’s **Properties** page and set the **Target framework** property to “.NET Framework 4.”

To open the **Properties** page, right-click the project in Solution Explorer and click **Properties** on the popup menu.

**Note**: If **EbiWuBlastHandler** is not defined, this is a good indication that you need to set the target framework.

5. Build the application.

6. Press CTRL+F5 to run the application.

BlastRequest uses a simple sequence that is defined within the application, but you can easily modify the application to load sequences from files. For an example of how to load data from files, see the [AlignSequences](#_How_to_Align) sample earlier in this document.

## BlastRequest Sample

Listing 3 is a complete listing of the BlastRequest sample. The numbered comments identify the key parts of the code and are discussed in notes that follow the listing.

Listing 3: BlastRequest

//[1]

using System;

using System.Collections.Generic;

using System.Threading;

using System.IO;

using Bio;

using Bio.IO;

using Bio.Registration;

using Bio.Web;

using Bio.Web.Blast;

namespace BlastRequest

{

class BlastRequest

{

static void Main(string[] args)

{

// [2] Prepare data

string seq = @"GACGCCGCCGCCACCACCGCCACCGCCGCAGCAGAAGCAGCGCACCGCAGGAGGGAAG" +

"ATGCCGGCGGGGCACGGGCTGCGGGCGCGGACGGCGACCTCTTCGCGCGGCCGTTCCGCAAGAAGGGTTA" +

"CATCCCGCTCACCACCTACCTGAGGACGTACAAGATCGGCGATTACGTNGACGTCAAGGTGAACGGTG";

Sequence sequence = new Sequence(Alphabets.DNA, seq);

// [3] Create and configure service handler

EbiWuBlastHandler blastService = new EbiWuBlastHandler();

ConfigParameters configParams = new ConfigParameters();

configParams.UseBrowserProxy = true;

blastService.Configuration = configParams;

// [4] Define query.

BlastParameters searchParams = new BlastParameters();

searchParams.Add("Program", "blastn");

searchParams.Add("Database", "em\_rel");

searchParams.Add("Expect", "1e-10");

searchParams.Add("Email", "*YourAddress*@*YourInstitution*");

// [5] create and submit request

string jobID;

try

{

jobID = blastService.SubmitRequest(sequence, searchParams);

}

catch

{

Console.WriteLine("Service is not available.");

return;

}

// [6] Wait for Ready status

ServiceRequestInformation info = blastService.GetRequestStatus(jobID);

if (info.Status != ServiceRequestStatus.Waiting

&& info.Status != ServiceRequestStatus.Ready)

{

Console.WriteLine("Service is not ready or waiting.");

return;

}

int maxAttempts = 10;

int attempt = 1;

while (attempt <= maxAttempts

&& info.Status != ServiceRequestStatus.Error

&& info.Status != ServiceRequestStatus.Ready)

{

++attempt;

info = blastService.GetRequestStatus(jobID);

Thread.Sleep(

info.Status == ServiceRequestStatus.Waiting

|| info.Status == ServiceRequestStatus.Queued

? 20000 \* attempt

: 0);

}

// [7] Get results

IList<BlastResult> results2 =

blastService.FetchResultsSync(jobID, searchParams) as List<BlastResult>;

}

}

}

## BlastRequest Notes

Although BlastRequest is quite simple, it shows how to use some of the key API elements and demonstrates a programming pattern that is used by many Framework Web service applications. The following list—which is keyed to the numbered comments in Listing 3—briefly describes the associated code. The sections following these notes provide a more detailed examination of the key topics.

### [1] Add using Statements for .NET Bio Framework Namespaces

In addition to the **Bio** namespace, Web service applications usually include **using** statements for:

* **Bio.Web**, which contains types that are used by all Bio Web service applications.
* The namespace that contains the types for the particular Web service, in this case, **Bio.Web.Blast**.

### [2] Prepare data for submission

For simplicity, BlastRequest creates a simple DNA sequence internally. In general, you prepare a sequence for submission based on your data and the requirements of the service.

### [3] Create and configure a service handler

.NET Bio Framework provides a service handler for each supported Web site. The handler for the EBI Blast service is **EbiWuBlastHandler**.

To configure the service handler

1. Create a Bio.Web.ConfigParameters object.

2. Set the configuration properties, as appropriate for the service.

BlastRequest sets UseBrowserProxy property to true, which specifies the default browser proxy settings.

For information on other configuration settings, see the .NET Bio Help file.

3. Assign the ConfigParameters object to the BLAST service handler’s **Configuration** property.

### [4] Define the query

To define the query, create a **BlastParameters** object, which is a container for a set of key-value pairs that specify the query parameters. To configure the object, use the **Add** method to add appropriate key-value pairs. Some of these values are generic and others are specific to a particular BLAST service. For more information, see the Web site for the particular service.

**Tip:** Values are often case sensitive, so make sure that you use the correct case. For example, the correct Program value for the NCBI BLAST service is “BLaStN,” which would not work for the EBI BLAST service.

BlastRequest sets the **Program**, **Database**, **Expect**, and **Email** parameters with values that are appropriate for the EBI BLAST service. The following table contains the complete list of keys, most of which are optional. For details on how to set the values, see the Web site that supports the particular service.

EBI BLAST Parameters

|  |  |
| --- | --- |
| Key | Value |
| **Alignments** | Number of alignments to return. |
| **Command** | Command to execute. |
| **CompositionBasedStatistics** | Type of composition based statistics to apply. |
| **Database** | Database name. |
| **EffectiveSearchSpace** | Effective length of the search space. |
| **Email** | Email address for reporting job problems. |
| **EntrezQuery** | Entrez query to limit the search. |
| **Expect** | Expect value. Higher values return more results. |
| **ExpectHigh** | Expect higher threshold for formatting. |
| **ExpectLow** | Expect lower threshold for formatting. |
| **Filter** | Sequence filter identifier. |
| **FormatType** | Type of data to return. |
| **GapCosts** | Gap open and gap extend costs. |
| **GeneticCode** | Query genetic code. |
| **HitlistSize** | Number of hits to keep. |
| **IThreshold** | Threshold for extending hits (PSI BLAST only). |
| **LowercaseMask** | Enable masking of lower case in query. |
| **MatrixName** | Matrix name (protein search only). |
| **NucleotideMatchReward** | Reward for a nucleotide match (blastn only). |
| **NucleotideMismatchPenalty** | Penalty for a nucleotide mismatch (blastn only). |
| **PhiPattern** | Phi Blast pattern. |
| **Program** | Program name. |
| **Pssm** | PSI BLAST checkpoint. |
| **Query** | Query sequence. |
| **QueryBelieveDefline** | Whether to believe **defline** in FASTA query. |
| **QueryFrom** | Start of subsequence (one offset). |
| **QueryTo** | End of subsequence (one offset)—zero means ignore. |
| **RID** | Identifier for stored request. |
| **Sensitivity** | Search sensitivity setting. |
| **Service** | Blast service which needs to be performed. |
| **Strand** | Which strand of DNA should be searched. |
| **Threshold** | Threshold for extending hits. |
| **UngappedAlignment** | Whether to perform an ungapped alignment. |
| **WordSize** | Word size - default is 3 for proteins, 11 for nuc-nuc, 28 for megablast. |

### [5] Create and submit a request

To submit a request, call the service handler’s **SubmitRequest** method, and pass it the sequence to be analyzed and the **BlastParameters** object that specifies the query.

**Important:** If the service is not available, **SubmitRequest** might throw an exception. We recommend wrapping this method call in a try-catch block so you can handle the exception appropriately. BlastRequest simply prints a message, but you could also implement a response such as running the query on a different service. Other Web service requests can also throw exceptions.

For brevity, BlastRequest wraps only the first request, but you should consider wrapping the other requests in **try-catch** blocks as well.

### [6] Wait for Ready status

Your job must have a **Ready** status before you can retrieve your results. To determine the job status, call the service handler’s **GetRequestStatus** method. If you have a valid job, the returned **ServiceRequestInformation.Status** property should be set to **ServiceRequestStatus.Ready** or **ServiceRequestStatus.Waiting**. Otherwise, you must resubmit the request.

It is quite common for the initial request to return a **Waiting** status. In that case, repeat the request until you are successful. BlastRequest in Listing 2 shows a simple example of how to submit repeated requests.

### [7] Get the results

After you receive a Ready status, you can retrieve the results by calling the service handler’s **FetchSyncResults** method, which returns a list of **Bio.Web.Blast.BlastResult** objects, one for each region of similarity that the service identifies.

Note: **FetchSyncResults** does not return until it has retrieved all the results from the site, so this step might take a significant amount of time.

# Extending .NET Bio Framework: How to Register Add-in Components

Users can implement a variety of custom components, including parsers, formatters, aligners, Web service connectors, and so on. If you register the component, the Framework automatically recognizes it at runtime, and exposes it to users along with other components of the same type. For example, if you implement and register a custom sequence parser, the **FindParserByFileName** method returns an instance of your parser when you request a parser for the associated file type.

The basic registration model is:

1. Implement a component, such as a parser or aligner, that exports a registrable interface.

2. Apply the Bio **RegistrableAttibute** attribute to the class.

3. Place the component DLL in a standard Add-ins folder.

The Framework scans the **Add-ins** folder for registered components and uses reflection to load the assemblies. The Framework then exposes the components to users along with all other components of the same type. This section describes how to register a component.

The following table lists the registrable components, and the associated interface.

Registrable Components

|  |  |
| --- | --- |
| Component | Interface |
| Alphabet | **IAlphabet** |
| Formatter | **IFormatter** |
| Parser | **IParser** |
| Sequence Aligner | **ISequenceAligner** |
| Sequence Assembler | **IDeNovoAssembler** |
| Web Service Handler | **IServiceHandler** |

To make a component registrable

1. Apply the **Bio.Registration.RegistrableAttribute** attribute to the class, and set it to **true**.

The following example shows how to make a custom parser registrable.

[RegistrableAttribute (true)]

public class MyCustomParser : ISequenceParser

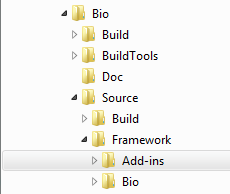
{

...

}

**Note:** In this example, **MyCustomParser** supports **IParser** indirectly, through **ISequenceParser**.

1. To have the Framework load the component, put the component’s DLL in the **Bio\Framework\Source\Add-Ins** folder. The following figure shows a typical location for this folder.



# Resources

This section provides links to additional information about .NET Bio Framework and related topics.

#### CodePlex Resources

.NET Bio Framework

<http://bio.codeplex.com/>  
Overview.docx  
Programming\_Guide.docx  
PaDeNA.docx  
Sequence Assembler User Guide.docx

Extension User Guide.docx

Sandcastle - Documentation Compiler for Managed Class Libraries

<http://sandcastle.codeplex.com/>

Sandcastle Help File Builder

<http://shfb.codeplex.com/>

#### Microsoft Biology Foundation Resources from Microsoft Research

Microsoft Biology Foundation at Microsoft Research

<http://research.microsoft.com/en-US/projects/bio/training.aspx>

#### Microsoft Resources

IronPython

<http://www.codeplex.com/IronPython/>

Microsoft DreamSpark

<https://www.dreamspark.com/default.aspx>

MSDN Academic Alliance

<http://msdn.microsoft.com/en-us/academic/default.aspx>

Visual Studio 2010 Testing the Application

[http://visualstudiogallery.msdn.microsoft.com/en-us/db5872a6-e5e5-4c1f-ad1c-a05cf2c143dc](http://msdn.microsoft.com/en-us/library/ms182409.aspx)

Project Trident: A Scientific Workflow Workbench

<http://research.microsoft.com/en-us/collaboration/tools/trident.aspx>

Visual Studio 2010 and .NET Framework 4

<http://msdn.microsoft.com/vstudio/>

#### Other Resources

Testing the Application using Visual Studio 2010 test features

<http://msdn.microsoft.com/en-us/library/ms182409.aspx>

Windows Installer XML (WiX) toolset

<http://wix.sourceforge.net/>

#### Bioinformatics References

BED format

http://genome.ucsc.edu/FAQ/FAQformat#format1

BLAST

<http://blast.ncbi.nlm.nih.gov/Blast.cgi>

EBI BLAST Service

<http://www.ebi.ac.uk/Tools/blast2/index.html>

FASTA format description

<http://www.ncbi.nlm.nih.gov/blast/fasta.shtml>

FASTQ format description

<http://maq.sourceforge.net/fastq.shtml>

GenBank

Overview:  
<http://www.ncbi.nlm.nih.gov/Genbank/>  
Sample GenBank Record: <http://www.ncbi.nlm.nih.gov/Sitemap/samplerecord.html>

GFF Specification

<http://www.sanger.ac.uk/resources/software/gff/spec.html>

International Nucleotide Sequence Database Collaboration

http://insdc.org

National Center for Biotechnology Information

http://www.ncbi.nlm.nih.gov

Phylogenetic tree

<http://en.wikipedia.org/wiki/Phylogenetic_tree>

Single-nucleotide polymorphism

<http://en.wikipedia.org/wiki/Single_nucleotide_polymorphism>

# Appendix A: Sample GenBank Data File

This appendix contains example data that you can use with the AlignSequences sample application. The data and metadata are based on the *Saccharomyces cerevisiae* gene sample from the GenBank Web site.

To keep the AlignSequences output to a manageable length, the sequence is truncated to a few hundred nucleotides. If you prefer to work with the complete sequence, you can obtain it from the GenBank Web site at <http://www.ncbi.nlm.nih.gov/nuccore/1293613>.

## GenBankSample1.gbk File

The following example is the data for the GenBankSample1.gbk file.

LOCUS SCU49845 5028 bp DNA PLN 21-JUN-1999

DEFINITION Saccharomyces cerevisiae TCP1-beta gene, partial cds, and Axl2p

(AXL2) and Rev7p (REV7) genes, complete cds.

ACCESSION U49845

VERSION U49845.1 GI:1293613

KEYWORDS .

SOURCE Saccharomyces cerevisiae (baker's yeast)

ORGANISM Saccharomyces cerevisiae

Eukaryota; Fungi; Ascomycota; Saccharomycotina; Saccharomycetes;

Saccharomycetales; Saccharomycetaceae; Saccharomyces.

REFERENCE 1 (bases 1 to 5028)

AUTHORS Torpey,L.E., Gibbs,P.E., Nelson,J. and Lawrence,C.W.

TITLE Cloning and sequence of REV7, a gene whose function is required for

DNA damage-induced mutagenesis in Saccharomyces cerevisiae

JOURNAL Yeast 10 (11), 1503-1509 (1994)

PUBMED 7871890

REFERENCE 2 (bases 1 to 5028)

AUTHORS Roemer,T., Madden,K., Chang,J. and Snyder,M.

TITLE Selection of axial growth sites in yeast requires Axl2p, a novel

plasma membrane glycoprotein

JOURNAL Genes Dev. 10 (7), 777-793 (1996)

PUBMED 8846915

REFERENCE 3 (bases 1 to 5028)

AUTHORS Roemer,T.

TITLE Direct Submission

JOURNAL Submitted (22-FEB-1996) Terry Roemer, Biology, Yale University, New

Haven, CT, USA

FEATURES Location/Qualifiers

source 1..5028

/organism="Saccharomyces cerevisiae"

/db\_xref="taxon:4932"

/chromosome="IX"

/map="9"

CDS <1..206

/codon\_start=3

/product="TCP1-beta"

/protein\_id="AAA98665.1"

/db\_xref="GI:1293614"

/translation="SSIYNGISTSGLDLNNGTIADMRQLGIVESYKLKRAVVSSASEA

AEVLLRVDNIIRARPRTANRQHM"

gene 687..3158

/gene="AXL2"

CDS 687..3158

/gene="AXL2"

/note="plasma membrane glycoprotein"

/codon\_start=1

/function="required for axial budding pattern of S.

cerevisiae"

/product="Axl2p"

/protein\_id="AAA98666.1"

/db\_xref="GI:1293615"

/translation="MTQLQISLLLTATISLLHLVVATPYEAYPIGKQYPPVARVNESF

TFQISNDTYKSSVDKTAQITYNCFDLPSWLSFDSSSRTFSGEPSSDLLSDANTTLYFN

VILEGTDSADSTSLNNTYQFVVTNRPSISLSSDFNLLALLKNYGYTNGKNALKLDPNE

VFNVTFDRSMFTNEESIVSYYGRSQLYNAPLPNWLFFDSGELKFTGTAPVINSAIAPE

TSYSFVIIATDIEGFSAVEVEFELVIGAHQLTTSIQNSLIINVTDTGNVSYDLPLNYV

YLDDDPISSDKLGSINLLDAPDWVALDNATISGSVPDELLGKNSNPANFSVSIYDTYG

DVIYFNFEVVSTTDLFAISSLPNINATRGEWFSYYFLPSQFTDYVNTNVSLEFTNSSQ

DHDWVKFQSSNLTLAGEVPKNFDKLSLGLKANQGSQSQELYFNIIGMDSKITHSNHSA

NATSTRSSHHSTSTSSYTSSTYTAKISSTSAAATSSAPAALPAANKTSSHNKKAVAIA

CGVAIPLGVILVALICFLIFWRRRRENPDDENLPHAISGPDLNNPANKPNQENATPLN

NPFDDDASSYDDTSIARRLAALNTLKLDNHSATESDISSVDEKRDSLSGMNTYNDQFQ

SQSKEELLAKPPVQPPESPFFDPQNRSSSVYMDSEPAVNKSWRYTGNLSPVSDIVRDS

YGSQKTVDTEKLFDLEAPEKEKRTSRDVTMSSLDPWNSNISPSPVRKSVTPSPYNVTK

HRNRHLQNIQDSQSGKNGITPTTMSTSSSDDFVPVKDGENFCWVHSMEPDRRPSKKRL

VDFSNKSNVNVGQVKDIHGRIPEML"

gene complement(3300..4037)

/gene="REV7"

CDS complement(3300..4037)

/gene="REV7"

/codon\_start=1

/product="Rev7p"

/protein\_id="AAA98667.1"

/db\_xref="GI:1293616"

/translation="MNRWVEKWLRVYLKCYINLILFYRNVYPPQSFDYTTYQSFNLPQ

FVPINRHPALIDYIEELILDVLSKLTHVYRFSICIINKKNDLCIEKYVLDFSELQHVD

KDDQIITETEVFDEFRSSLNSLIMHLEKLPKVNDDTITFEAVINAIELELGHKLDRNR

RVDSLEEKAEIERDSNWVKCQEDENLPDNNGFQPPKIKLTSLVGSDVGPLIIHQFSEK

LISGDDKILNGVYSQYEEGESIFGSLF"

ORIGIN

1 gatcctccat atacaacggt atctccacct caggtttaga tctcaacaac ggaaccattg

61 ccgacatgag acagttaggt atcgtcgaga gttacaagct aaaacgagca gtagtcagct

121 ctgcatctga agccgctgaa gttctactaa gggtggataa catcatccgt gcaagaccaa

181 gaaccgccaa tagacaacat atgtaacata tttaggatat acctcgaaaa taataaaccg

241 ccacactgtc attattataa ttagaaacag aacgcaaaaa ttatccacta tataattcaa

//

## GenBankSample2.gbk File

The following example includes a slightly modified version of the sequence data from GenBankSample1.gbk, with the original metadata. You can modify the metadata, if you prefer.

LOCUS SCU49845 5028 bp DNA PLN 21-JUN-1999

DEFINITION Saccharomyces cerevisiae TCP1-beta gene, partial cds, and Axl2p

(AXL2) and Rev7p (REV7) genes, complete cds.

ACCESSION U49845

VERSION U49845.1 GI:1293613

KEYWORDS .

SOURCE Saccharomyces cerevisiae (baker's yeast)

ORGANISM Saccharomyces cerevisiae

Eukaryota; Fungi; Ascomycota; Saccharomycotina; Saccharomycetes;

Saccharomycetales; Saccharomycetaceae; Saccharomyces.

REFERENCE 1 (bases 1 to 5028)

AUTHORS Torpey,L.E., Gibbs,P.E., Nelson,J. and Lawrence,C.W.

TITLE Cloning and sequence of REV7, a gene whose function is required for

DNA damage-induced mutagenesis in Saccharomyces cerevisiae

JOURNAL Yeast 10 (11), 1503-1509 (1994)

PUBMED 7871890

REFERENCE 2 (bases 1 to 5028)

AUTHORS Roemer,T., Madden,K., Chang,J. and Snyder,M.

TITLE Selection of axial growth sites in yeast requires Axl2p, a novel

plasma membrane glycoprotein

JOURNAL Genes Dev. 10 (7), 777-793 (1996)

PUBMED 8846915

REFERENCE 3 (bases 1 to 5028)

AUTHORS Roemer,T.

TITLE Direct Submission

JOURNAL Submitted (22-FEB-1996) Terry Roemer, Biology, Yale University, New

Haven, CT, USA

FEATURES Location/Qualifiers

source 1..5028

/organism="Saccharomyces cerevisiae"

/db\_xref="taxon:4932"

/chromosome="IX"

/map="9"

CDS <1..206

/codon\_start=3

/product="TCP1-beta"

/protein\_id="AAA98665.1"

/db\_xref="GI:1293614"

/translation="SSIYNGISTSGLDLNNGTIADMRQLGIVESYKLKRAVVSSASEA

AEVLLRVDNIIRARPRTANRQHM"

gene 687..3158

/gene="AXL2"

CDS 687..3158

/gene="AXL2"

/note="plasma membrane glycoprotein"

/codon\_start=1

/function="required for axial budding pattern of S.

cerevisiae"

/product="Axl2p"

/protein\_id="AAA98666.1"

/db\_xref="GI:1293615"

/translation="MTQLQISLLLTATISLLHLVVATPYEAYPIGKQYPPVARVNESF

TFQISNDTYKSSVDKTAQITYNCFDLPSWLSFDSSSRTFSGEPSSDLLSDANTTLYFN

VILEGTDSADSTSLNNTYQFVVTNRPSISLSSDFNLLALLKNYGYTNGKNALKLDPNE

VFNVTFDRSMFTNEESIVSYYGRSQLYNAPLPNWLFFDSGELKFTGTAPVINSAIAPE

TSYSFVIIATDIEGFSAVEVEFELVIGAHQLTTSIQNSLIINVTDTGNVSYDLPLNYV

YLDDDPISSDKLGSINLLDAPDWVALDNATISGSVPDELLGKNSNPANFSVSIYDTYG

DVIYFNFEVVSTTDLFAISSLPNINATRGEWFSYYFLPSQFTDYVNTNVSLEFTNSSQ

DHDWVKFQSSNLTLAGEVPKNFDKLSLGLKANQGSQSQELYFNIIGMDSKITHSNHSA

NATSTRSSHHSTSTSSYTSSTYTAKISSTSAAATSSAPAALPAANKTSSHNKKAVAIA

CGVAIPLGVILVALICFLIFWRRRRENPDDENLPHAISGPDLNNPANKPNQENATPLN

NPFDDDASSYDDTSIARRLAALNTLKLDNHSATESDISSVDEKRDSLSGMNTYNDQFQ

SQSKEELLAKPPVQPPESPFFDPQNRSSSVYMDSEPAVNKSWRYTGNLSPVSDIVRDS

YGSQKTVDTEKLFDLEAPEKEKRTSRDVTMSSLDPWNSNISPSPVRKSVTPSPYNVTK

HRNRHLQNIQDSQSGKNGITPTTMSTSSSDDFVPVKDGENFCWVHSMEPDRRPSKKRL

VDFSNKSNVNVGQVKDIHGRIPEML"

gene complement(3300..4037)

/gene="REV7"

CDS complement(3300..4037)

/gene="REV7"

/codon\_start=1

/product="Rev7p"

/protein\_id="AAA98667.1"

/db\_xref="GI:1293616"

/translation="MNRWVEKWLRVYLKCYINLILFYRNVYPPQSFDYTTYQSFNLPQ

FVPINRHPALIDYIEELILDVLSKLTHVYRFSICIINKKNDLCIEKYVLDFSELQHVD

KDDQIITETEVFDEFRSSLNSLIMHLEKLPKVNDDTITFEAVINAIELELGHKLDRNR

RVDSLEEKAEIERDSNWVKCQEDENLPDNNGFQPPKIKLTSLVGSDVGPLIIHQFSEK

LISGDDKILNGVYSQYEEGESIFGSLF"

ORIGIN

1 gatcctccat rrrraacggt atctccacct caggtttaga tctcaacaac ggaaccattg

61 ccgacatgag acagttaggt atcgtcgaga gttacaagct aaaacgagca gtagtcagct

121 ctgcatctga agccgctgaa gttctactaa gggtggataa catcatccgt gcaagaccaa

181 gaaccgccaa tagacaacat atgtaacata tttaggatat acctcgaaaa taataaaccg

241 ccacactgtc attattataa ttagaaacag aacgcaaaaa ttatccacta tataattcaa

//

# Appendix B: RNA and Protein Alphabets

This appendix describes the .NET Bio Framework RNA and protein alphabets.

## The RNA Alphabet

**RnaAlphabet** contains a set of fields, each of which contains an **BIO.Nucleotide** object that represents one of the members of the alphabet. The following table lists the fields.

RNA Sequence Alphabets

|  |  |  |
| --- | --- | --- |
| Field | Symbol | Name |
| **A** | **A** | Adenine |
| **C** | **C** | Cytosine |
| **G** | **G** | Guanine |
| **U** | **U** | Uracil |
| **AC** | **M** | A or C |
| **ACU** | **H** | A, C, or U |
| **AU** | **W** | A or U |
| **GA** | **R** | G or A |
| **GAU** | **D** | G, A, or D |
| **GC** | **S** | G or C |
| **GCA** | **V** | G, C, or A |
| **GU** | **K** | G or U |
| **GUC** | **B** | G, U, or C |
| **UC** | **Y** | U or C |
| **Any** | **N** | A, C, T, or T |
| **Gap** | **-** | A gap |

## Protein Alphabet

**ProteinAlphabet** contains a set of fields, each of which contains a **Bio.AminoAcid** object that represents one of the members of the alphabet. The following table lists the fields.

RNA Sequence Alphabets

|  |  |  |
| --- | --- | --- |
| Field | Symbol | Name |
| **Ala** | **A** | Alanine |
| **Asx** | **B** | Aspartic Acid or Asparagine |
| **Cys** | **C** | Cysteine |
| **Asp** | **D** | Aspartic Acid |
| **Glu** | **E** | Glutamic Acid |
| **Phe** | **F** | Phenylalanine |
| **Gly** | **G** | Glycine |
| **His** | **H** | Histidine |
| **Ile** | **I** | Isoleucine |
| **Xle** | **J** | Leucine or Isoleucine |
| **Lys** | **K** | Lysine |
| **Leu** | **L** | Leucine |
| **Met** | **M** | Methionine |
| **Asn** | **N** | Asparagine |
| **Pyl** | **O** | Pyrrolysine |
| **Pro** | **P** | Proline |
| **Gln** | **Q** | Glutamine |
| **Arg** | **R** | Arginine |
| **Ser** | **S** | Serine |
| **Thr** | **T** | Threoine |
| **Sel** | **U** | Selenocysteine |
| **Val** | **V** | Valine |
| **Trp** | **W** | Tryptophan |
| **Tyr** | **Y** | Tyrosine |
| **Glx** | **Z** | Glutamic Acid or Glutamine |
| **Xxx** | **X** | Undetermined or atypical |
| **Term** | **\*** | Termination |
| **Gap** | **---** | Gap |