|  |  |
| --- | --- |
|  | Microsoft Biology Foundation Programming Guide  Version 1.0 - June 2010 |

Abstract

The Microsoft® Biology Foundation (MBF) 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 MBF applications.

Microsoft Biology Foundation software and documentation are available at: <http://mbf.codeplex.com>

In this document

[Introduction 4](#_Toc264901702)

[Terminology 4](#_Toc264901703)

[Getting Started 6](#_Toc264901704)

[An MBF Quick Start 9](#_Toc264901705)

[MBF Architecture 13](#_Toc264901706)

[Input and Output: Parsers and Formatters 15](#_Toc264901707)

[Input and Output: Web Service Connectors 20](#_Toc264901708)

[Object Model: Sequences and Related Types 20](#_Toc264901709)

[Object Model: Other Types 25](#_Toc264901710)

[Data Processing: Algorithms 27](#_Toc264901711)

[Example: How to Manipulate a Sequence 29](#_Toc264901712)

[Example: How to Submit an MBF Web Services Request 32](#_Toc264901713)

[Extending MBF: How to Register Add-in Components 37](#_Toc264901714)

[Extending MBF: Data Virtualization 38](#_Toc264901715)

[Resources 41](#_Toc264901716)

[Appendix A: Sample GenBank Data File 43](#_Toc264901717)

[Appendix B: RNA and Protein Alphabets 46](#_Toc264901718)

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

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

© 2010 Microsoft Corporation. All rights reserved.

Microsoft, 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](#_Toc264959916)

[Terminology 4](#_Toc264959917)

[Getting Started 6](#_Toc264959918)

[Installation 6](#_Toc264959919)

[Prerequisites 6](#_Toc264959920)

[Hardware and Software Requirements 7](#_Toc264959921)

[Start a New MBF Project 7](#_Toc264959922)

[MBFWorkflow Activities for Project Trident 9](#_Toc264959923)

[An MBF Quick Start 9](#_Toc264959924)

[AlignSequences Sample Application 11](#_Toc264959925)

[AlignSequences Notes 12](#_Toc264959926)

[MBF Architecture 13](#_Toc264959927)

[Applications 14](#_Toc264959928)

[I/O and Analysis 14](#_Toc264959929)

[Object Model 15](#_Toc264959930)

[Input and Output: Parsers and Formatters 15](#_Toc264959931)

[Parsers 17](#_Toc264959932)

[Formatters 19](#_Toc264959933)

[Encoding Data 19](#_Toc264959934)

[Input and Output: Web Service Connectors 20](#_Toc264959935)

[Object Model: Sequences and Related Types 20](#_Toc264959936)

[Alphabets 21](#_Toc264959937)

[The Sequence Object 22](#_Toc264959938)

[Sequence Manipulation 24](#_Toc264959939)

[The SequenceRange Object 25](#_Toc264959940)

[The AlignedSequence Object 25](#_Toc264959941)

[Object Model: Other Types 25](#_Toc264959942)

[Phylogenetics 25](#_Toc264959943)

[SNP Items 26](#_Toc264959944)

[MBF.Matrices 26](#_Toc264959945)

[Data Processing: Algorithms 27](#_Toc264959946)

[Example: How to Manipulate a Sequence 29](#_Toc264959947)

[SequenceManipulation Notes 30](#_Toc264959948)

[Example: How to Submit an MBF Web Services Request 32](#_Toc264959949)

[BlastRequest Sample 33](#_Toc264959950)

[BlastRequest Notes 34](#_Toc264959951)

[Extending MBF: How to Register Add-in Components 37](#_Toc264959952)

[Extending MBF: Data Virtualization 38](#_Toc264959953)

[Data Virtualization Architecture 38](#_Toc264959954)

[How to Implement Data Virtualization 39](#_Toc264959955)

[Resources 41](#_Toc264959956)

[Appendix A: Sample GenBank Data File 43](#_Toc264959957)

[GenBankSample1.gbk File 43](#_Toc264959958)

[GenBankSample2.gbk File 44](#_Toc264959959)

[Appendix B: RNA and Protein Alphabets 46](#_Toc264959960)

[The RNA Alphabet 46](#_Toc264959961)

[Protein Alphabet 47](#_Toc264959962)

# Introduction

The Microsoft Biology Foundation (MBF) is an open source, reusable .NET Framework library and application programming interface (API) for bioinformatics research. Application developers can use MBF 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.

MBF represents sequence data and metadata with format-independent **Sequence** or **SequenceRange** 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 MBF environment.

MBF 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 MBF add-in for Microsoft Office Excel. For details, see “Microsoft Research Biology 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 MBF applications in C#; other languages follow a very similar programming pattern.

# Terminology

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

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.

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.

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.

Protein

An 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.

SNP (single-nucleotide polymorphism)

Items represent sequence variations between species or paired chromosomes.

# Getting Started

This section describes basic system requirements and installation, and summarizes steps for starting an MBF project and building it.

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

## Installation

MBF is hosted on <http://mbf.codeplex.com/>. Application developers have two primary installation options:

* Synchronize to the MBF source code tree.

This option allows you to use and modify the MBF source under the [Microsoft Public License](http://opensource.org/licenses/ms-pl.html) (Ms-PL). You have immediate access to the latest changes and can also contribute code to the project. The MBF source tree is a single Visual Studio® 2010 solution, so you can build all MBF DLLs by loading Mbf.sln and running **Build Solution**. For details on how to become a contributor, download “Contribution Guide” from the site’s Documentation tab.

- OR -

* Run the MBF installer (MBF.msi) and follow the directions provided by the installation wizard.

The complete installation option installs everything that you need to implement MBF applications—including all MBF DLLs—under Program Files\Microsoft Biology Initiative\1.0\MBF or Program Files (x86)\Microsoft Biology Initiative\1.0\MBF, on x86 and x64 systems, respectively.

Both options include documentation and samples. For more details, see the MBF CodePlex Web site.

You can also download the Excel Add-in and the Sequence Assembler tool. For more details and download instructions, see the MBF 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 to implement MBF applications:

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

You must have the following software to implement MBF applications:

* Windows® XP SP3 or later, x86 or x64 versions
* Visual Studio 2010
* .NET Framework 4.0, which is included with Visual Studio 2010
* MBF 1.0 or later

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

Optional software includes:

* IronPython, if you want to use this language to implement MBF applications.
* Microsoft Project Trident, if you want to implement MBF workflow activities for Trident.
* NUnit, for creating and running unit test cases.

For more information, see “NUnit integration for Visual Studio” listed in Resources.

* Sandcastle and Sandcastle Help File Builder, if you want to build your own MBF API reference.

You must use the June 2010 or later releases of these applications to build the MBF API reference.

* WIX, for building setup installers.

For more information on these software packages, see “Resources” at the end of this document.

## Start a New MBF Project

The MBF 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.

### MBF Console Applications

For console applications, the simplest approach is to use the Visual Studio MBF Console Application template, which is installed with the MBF package. This template automatically references the appropriate DLLs and provides starting code.

To start a new MBF 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 **MBF Console Application** template, provide an appropriate name and location, and click **OK** to opens the **MBF 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. MBF Console Application includes the following operations:

* Pairwise 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 a PaDeNa Denovo assembler.
* Online Blast Service: Creates several method template 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.

### MBF 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 MBF template for GUI applications, but the following procedure describes how to set up a standard project for MBF.

To start a new GUI-based MBF project

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

2. Reference the following MBF DLLs:

(Required) MBF.dll, which contains the core MBF object model.

(Optional) MBF.WebServiceHandlers.dll, if you want to use the MBF 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.

## MBFWorkflow 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 doesn’t 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 (see “Resources” at the end of this document).

If you have downloaded the MBF source tree, you can see several examples of Trident bioinformatics activities under the Samples\MBF.Workflow directory.

# An MBF Quick Start

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

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

1. Read input sequences from storage and convert them to MBF 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.

If you have installed MBF, 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 MBF.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” 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” 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.”

## 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.

Listing 1: AlignSequences

//[1]

using System;

using System.Collections.Generic;

using MBF;

using MBF.IO.GenBank;

using MBF.IO.Fasta;

using MBF.Algorithms.Alignment;

using MBF.SimilarityMatrices;

namespace AlignSequences

{

class AlignSequences

{

static void Main(string[] args)

{

//[2]

GenBankParser parser = new GenBankParser();

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

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

//[3]

DnaAlphabet dna = DnaAlphabet.Instance;

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));

}

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

//Omitted: Print statistics for the second sequence

//[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);

}

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

Console.WriteLine("Second Sequence: {0}\n", item.SecondSequence.ToString());

Console.WriteLine("Consensus: {0}",

item.PairwiseAlignedSequences[0].Consensus);

}

//[6]

ISequence outputSequence = result[0].PairwiseAlignedSequences[0].Consensus;

FastaFormatter outputFormatter = new FastaFormatter();

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

}

}

}

## 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 MBF 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 MBF Namespaces

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

### [2] Read input data from storage

MBF 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 MBF object model.

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

### [3] Validate Input Data

AlignSequences checks for obvious problems by printing the count of each nucleotide in the sequence. **DnaAlphabet.LookupAll** returns a list that contains each nucleotide symbol in the DNA alphabet. AlignSequences uses this list and **ISequence.Statistics.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 MBF objects, you can use MBF 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

MBF includes a set of formatters that write the contents of a **Sequence** or **SequenceRange** object to an appropriately formatted file. MBF 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** or **SequenceItem** to represent data but returning an interface on the object is used throughout the MBF API. For more discussion of this pattern, see “Object Model: Alphabets, Sequences and Sequence Ranges” later in this document.

# MBF Architecture

Figure 1 shows the basic MBF architecture.



Figure 1. MBF architecture

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

## Applications

MBF includes two applications that use the underlying MBF infrastructure:

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

For download links for these applications, see the Microsoft Biology Foundation site on CodePlex.

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

MBF also supports several utility applications, including:

* ReadGenerator, which produces data in a short-read form, similar to what might be produced by a next-generation sequencing machine.
* SAMUtils are command-line tools that perform various operations on SAM and BAM-formatted files.

## I/O and Analysis

I/O and analysis components both operate on the MBF 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

MBF 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 MBF object model, a format-independent internal representation. Most parsers have a corresponding formatter that converts data from the MBF object model to the associated format and writes the data to storage.

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

Some data sets are too large to be loaded into system memory as a unit. To allow applications to work with arbitrarily large data sets, many parsers and formatters support data virtualization, which interacts with the object model to load only the subset of data that is currently in use.

Web Service connectors transmit MBF sequence data to a remote site for analysis and return the results to the application. MBF includes connectors for the NCBI, EBI, and BioHPC services. Users can extend MBF by implementing and registering Web Service connectors for other sites and services.

### Analysis Components

MBF 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” later in this guide. Users can extend MBF by implementing and registering custom tools and utilities.

**Caution:** The MBF 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 MBF tools and functionality with output from similar tools and functionality implemented for other platforms, which might not use 0-based indices.

## Object Model

MBF 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.

For very large data sets, the object model interacts with the appropriate parsers and formatters to support data virtualization.

# Input and Output: Parsers and Formatters

Sequence-related data is typically stored as plain text files in a variety of formats. MBF 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 MBF applications is to use a parser to read the data from storage and convert it to the MBF object model, such as **Sequence** or **SequenceRange** objects. Those objects can then be used by subsequent MBF operations.

Most parsers have a corresponding formatter that writes the data from the object model to storage in the appropriate format. Because MBF 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 MBF is to implement a format converter.

For very large data sets that cannot be loaded into memory as a unit, several of the parsers support data virtualization, which loads only a working subset of the data at any given time. For more details, see “Data Virtualization” later in this section. Applications typically use virtualization implicitly, by calling a virtualization-enabled parser and letting the parser decide whether to virtualize the data. They can also direct the parser to virtualize data by setting the parser’s **EnforceDataVirtualization** property.

By default, parsers store DNA and RNA data with NCBI4NA encoding and protein data with NCBIStdAA encoding. You can specify other encoding schemes by providing the parser or formatter with an appropriate encoding object when you create the object. For further discussion, see “Encoding Sequence Data” later in this paper.

Table 1 describes the standard parsers and formatters supported by MBF. 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 **MBF.IO.GenBank** namespace. The exception is **SnpParser**, which is in the **MBF.IO** namespace. Types that support all parsers and formatters are in the **MBF.IO** namespace.

Table 1. MBF Parsers and Formatters

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

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

* The BED parser exposes **ParseRange** and **ParseRangeGrouping** methods rather than **Parse** and **ParseOne**. The table lists the return value of **BedParser.ParseRange**.
* The Newick parser exposes only a **Parse** method, and returns a **MBF.Phylogenetics**.Tree object rather than an interface.

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

## Parsers

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

To use an MBF parser

1. Create a parser object for the input format.

2. Pass the file to the appropriate parsing method, along with any related information such as whether the returned object should be read-only.

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 don’t know the file format, try passing the file name to the **MBF.IO.SequenceParsers.FindParserByFile** 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.

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

* Most parsers support two standard input methods, **ParseOne** and **Parse**, which handle files that contain single and multiple sequences, respectively.

**ParseOne** returns a single interface and **Parse** returns a **List<...>** object containing a list of interfaces. 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, **TextReader** objects, and **MBFTextReader** objects.
* The input methods allow you to specify whether the returned interface is read-only.

If you want to modify the sequence, use a **Parse** or **ParseOne** overload with a *isReadOnly* parameter, and set it to **false** to direct the parser to return a writeable object.

* You can determine how the sequence data is to be encoded by specifying an encoder when you create the parser object.

### 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 MBF objects, and returns the appropriate MBF interfaces. The details are beyond the scope of this document, but if you download the MBF 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 MBF\IO folder.

### Data Virtualization

Data files are sometimes large enough that hardware limitations prevent a parser from loading the entire data set into memory. There are two basic scenarios:

* Large sequences.

The data set contains one or more sequences that are too large to be loaded as a unit.

* Large sets.

The individual sequences are small enough to be loaded as a unit, but the data set contains too many sequences to load all of them at once.

MBF data virtualization allows applications to work with either type of large data set. With data virtualization, is divided into blocks. As applications works through a sequence, the parser provides the data block by block, as required.

**Tip:** To determine whether a parser supports data virtualization, check whether the type supports **EnforceDataVirtualization** or **IsDataVirtualizationEnabled** properties. All data virtualization-enabled parsers must support these properties.

Applications can use data virtualization in two ways:

* Implicitly, by calling a virtualization-enabled parser and letting the parser decide whether to use virtualization.
* Explicitly, by setting the parser’s **EnforceDataVirtualization** property.

Data virtualization typically slows the application, so it should be enabled only when necessary.

For the most part, data virtualization is transparent to applications. You work with the MBF object model in exactly the same way as you would for non-virtualized data. The object model handles the mechanics of obtaining the data from the parser, as required.

**Caution:** MBF data virtualization supports files up to 2 GB in size. Attempting to work with larger files might lead to instability on computers with insufficient memory.

For a discussion of how to implement data virtualization for a custom parser, “How to Implement Data Virtualization” later in this guide.

## Formatters

Most parsers have a corresponding formatter, which writes the contents of one or more MBF 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.

## Encoding Data

Storing data as an array of characters is not very efficient, especially for long sequences, MBF objects store data in a more compact form. The default encoding is determined by the alphabet. For example, DNA sequences are encoded by default in NCBI4na format, which uses 16 values to represent the four nucleotides, common ambiguities found when sequencing DNA and RNA, and gaps. This format thus requires 4-bits per sequence element instead of an entire character.

MBF supports several encoding schemes. To use a non-default encoding, provide the parser with an encoder object that understands the encoding scheme by passing the encoder’s **IEncoding** interface to the parser object’s constructor.

MBF includes the encoders listed in Table 2.

Table 2. MBF Encoders

|  |  |
| --- | --- |
| Encoding | MBF type |
| IUPACna | **MBF**.IO.Encoding.IupacNAEncoding |
| NCBI2na | **MBF**.IO.Encoding.Ncbi2NAEncoding |
| NCBI4na | **MBF**.IO.Encoding.Ncbi4NAEncoding |
| NCBIeaa | **MBF**.IO.Encoding.NcbiEAAEncoding |
| NCBIstdaa | **MBF**.IO.Encoding.NcbiStdAAEncoding |

These encoders support the following mappings:

|  |  |
| --- | --- |
| DNA to IUPACna DNA to NCBI2na DNA to NCBI4na IUPACna to DNA IUPACna to RNA NCBI2na to DNA NCBI2na to RNA NCBI4na to DNA | NCBI4na to RNA NCBIeaa to protein NCBIstdaa to protein Protein to NCBIeaa Protein to NCBIstdaa RNA to IUPACna RNA to NCBI2na RNA to NCBI4na |

For more details, see the MBF Help file.

You can also implement a custom encoder, but the details are beyond the scope of this document. If you download the MBF sources, you can use the standard encoders as a guide. The source files are all in the MBF\Encoding folder.

# 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. MBF includes a Web service API that simplifies the process of submitting a sequence for analysis. The following is a general procedure for using the MBF Web service API to submit sequences to remote sites.

To submit data to a remote site

1. Create an MBF 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 an MBF Web Services Request” 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, MBF includes connectors for the following BLAST Web services. You can extend MBF by implementing and registering connectors for other Web Services.

Table 3. 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:** MBF 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

The **MBF.Sequence** and **MBF.SequenceRange** types are the core of the MBF API. They essentially an efficient format-independent way to store sequence data and metadata and represent that data in the MBF environment. Each sequence has an associated alphabet that defines the available symbols.

## 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.

MBF supports three alphabets:

Alphabet Represented by

DNA **MBF.DnaAlphabet**

RNA **MBF.RnaAlphabet**

Protein **MBF.ProteinAlphabet**

For simplicity, this document focuses primarily on the DNA alphabet. The RNA and Protein alphabets are listed in Appendix B.

**DnaAlphabet** contains a set of fields, each of which contains a **MBF.Nucleotide** object that represents one of the members of the alphabet. **Nucleotide** supports **ISequenceItem**, so an MBF DNA sequence is essentially an array of **Nucleotide** objects.

In practice, you usually interact with **Nucleotide** objects through **ISequenceItem**, which exposes all of the key **Nucleotide** properties, including:

* **Name**: A friendly name, such as Adenine.
* **Symbol**: A standard IUPAC symbol, such as A or M.
* **Value**: The encoded element value.

**DnaAlphabet** has a set of public fields that contain **Nucleotide** objects for the supported values, which are listed in Table 4.

Table 4. DNA Sequence Alphabets

|  |  |  |
| --- | --- | --- |
| Field | Symbol | Name |
| **A** | **A** | Adenine |
| **C** | **C** | Cytosine |
| **G** | **G** | Guanine |
| **T** | **T** | Thymine |
| **AC** | **M** | A or C |
| **ACT** | **H** | A, C, or T |
| **Any** | **N** | A, C, T, or T |
| **AT** | **W** | A or T |
| **GA** | **R** | G or A |
| **Gap** | **-** | A gap |
| **GAT** | **D** | G, A, or T |
| **GC** | **S** | G or C |
| **GCA** | **V** | G, C, or A |
| **GT** | **K** | G or T |
| **GTC** | **B** | G, T, or C |
| **TC** | **Y** | T or C |

## 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. In particular, **ISequence** does not expose a setter for the **Sequence**.**IsReadOnly** property. If that property is set to **true** on the underlying object, **ISequence** is read-only and you cannot use it to modify the underlying object. You must explicitly direct the parser to return a writeable interface, as described earlier in “Input and Output: Parsers and Formatters”.

Each sequence element is represented by a **Nucleotide** object from **DnaAlphabet**. **Sequence** is an indexed object, which allows you to enumerate the nucleotides in a DNA sequence as if the object were an array of type **ISequenceItem**.

### 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. Applications typically obtain **Sequence** objects—or more accurately, their **ISequence** interfaces—from parsers. However, you can also create **Sequence** objects from scratch.

**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.

* Optionally, an encoding object.

If you don’t specify an encoding object, **Sequence** uses NCBI4na encoding.

The following example creates a **Sequence** object for a simple DNA sequence, GATTCCA. For simplicity, the example uses a string literal and explicitly specifies **DnaAlphabet**.

Sequence mySequence = new Sequence(DnaAlphabet, "GATTCCA");

Because **Sequence** is indexed, you can also construct a sequence by using **Sequence.Add** to add nucleotides one at a time, as follows:

Sequence mySequence = new Sequence(Alphabets.Dna);

mySequence.IsReadOnly = false;

mySequence.Add(Alphabets.Dna.G);

mySequence.Add(Alphabets.Dna.A);

mySequence.Add(Alphabets.Dna.T);

...

### How to Enumerate a Sequence

The simplest way to enumerate a sequence is with **foreach**, as shown by the following snippet from AlignSequences:

foreach (ISequenceItem item in nucList)

{

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

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

}

### 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 Table 5.

Table 5. Sequence Metadata Properties

|  |  |
| --- | --- |
| Property | Description |
| **ID** | An identifier. ID is usually just a brief code to distinguish this sequence from others. |
| **DisplayID** | A “friendly” ID, typically a readable string that describes the sample. |
| **MoleculeType** | The molecule type: DNA, RNA, protein, and so on. |

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. MBF 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:

BasicDerivedSequence

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.

Qualitative Sequence

Represents sequence data with quality scores. It is the basis for the FASTQ format.

SegmentedSequence

Represents a sequence that is constructed from several smaller sequences, one following another. It allows you to define a larger sequence from several smaller sequences without duplicating the memory usage.

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.

VirtualSequence

Represents just sequence metadata. It is typically used when you are interested in only the metadata.

For more information on these objects, see the MBF Help file.

## Sequence Manipulation

The **ISequence** interface includes methods and properties that you can use to manipulate sequences in various ways. These methods work only if **ISequence** is writeable, so you should first verify that **ISequence.IsReadOnly** is set to **false**.

Table 6 lists the relevant methods, and Table 7 lists the relevant properties.

Table 6. Sequence Manipulation Methods

|  |  |
| --- | --- |
| Method | Description |
| **Insert** | Inserts an item into the sequence. |
| **InsertRange** | Inserts a group of items into a sequence. |
| **RemoveRange** | Removes a group of items from a sequence. |
| **Replace** | Replaces an item in a sequence with a specified item. |
| **ReplaceRange** | Replaces a group of items in a sequence with a specified group. |

Table 7. Sequence Manipulation Properties

|  |  |
| --- | --- |
| Property | Description |
| **Complement** | Returns an **ISequence** interface on a **BasicDerivedSequence** object that represents the complement of the original sequence. |
| **Reverse** | Returns an **ISequence** interface on a **BasicDerivedSequence** object that represents the reverse of the original sequence. |
| **ReverseComplement** | Returns an **ISequence** interface on a **BasicDerivedSequence** object that represents the reverse complement of the original sequence. |

For an example of how to use these methods and properties, see “Example: How to Manipulate a Sequence” later in this paper.

## The SequenceRange Object

Most of the formats that MBF supports describe a complete sequence. However, it is also useful to represent genomic interval data rather than explicit sequences. MBF uses **SequenceRange** objects or the **ISequenceRange** interface to represent genomic intervals. In particular, MBF 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 often encountered through the **ISequenceAlignment** interface, which is basically a container for a list of **IAlignedSequence** interfaces.

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

# Object Model: Other Types

The MBF 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. Figure 2 shows a schematic version of a simple phylogenetic tree.



Figure 2. Phylogenetic tree

The MBF object model represents trees as follows:

* A tree is represented as a **MBF.Phylogenetics.Tree** object, which contains the tree’s root node.
* A node is represented as a **MBF.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 **MBF.Phylogenetics.Edge** object, which has a **Distance** property that contains the edge length.

The Newick parser, **MBF.IO.Newick.NewickParser**, reads phylogenetic trees stored as Newick-formatted files and returns a **MBF.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 represent sequence variations between species or paired chromosomes. MBF 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.

## MBF.Matrices

Bioinformatics uses matrices in a variety of ways. The **MBF.Matrix** namespace provides general-purpose support for matrix-related techniques. Figure 3 shows a schematic representation of an MBF matrix.



Figure 3. MBF 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 **MBF.Matrix** namespace in the MBF Help File.

# Data Processing: Algorithms

After you have created one or more sequence objects, you can manipulate or analyze the sequence data in various ways.

* The **Sequence** object supports methods and properties that you can use to manipulate sequences.
* **MBF.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 **MBF.Algorithms** and its child namespaces. For details, see the MBF 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.

#### MBF.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.

#### MBF.Algorithms.Alignment Namespace

This namespace includes a collection of types that support a variety of standard sequence alignment algorithms, as summarized in Table 8.

Table 8. Sequence Alignment Types

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

#### MBF.Algorithms.Assembly

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

#### MBF.Algorithms.Assembly.Graph

This namespace supports de Bruijn graphs.

#### MBF.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.

#### MBF.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.

#### MBF.Algorithms.Translation

This namespace contains several types that support translation, as listed in Table 9.

Table 9. 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 MBF to manipulate a sequence, including:

* Extract a segment from a larger sequence.
* Generate reverse, complement, and reverse complement sequences.
* Modify symbols.
* Add or delete 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 “[An MBF Quick Start](#_An_MBF_Quick)” earlier in this guide.

Listing 2: 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 output is:

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

## SequenceManipulation Notes

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

### [1] Create a Writeable Sequence Object

For simplicity, this example creates a sequence by using a string literal. Typically, you would use a parser to read the sequence from a data file.

**Sequence** objects are read-only by default. To manipulate a sequence, you must explicitly make the **Sequence** object writeable by setting its **IsReadOnly** property to **false**.

* If you create a sequence from scratch—as with this example—set **IsReadOnly** to **false**.
* If you read a sequence from a data file, you must explicitly direct the parser to return a writeable **ISequence** interface.

**ISequence.IsReadOnly** is a read-only property and must be set by the parser. By default, parsers set **ISequence.IsReadOnly** to **true**. For details about how to request a writeable **ISequence** interface, see “Parsers” earlier in this paper.

### [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 by calling the **Sequence** object’s **Range** method, and provide the segment’s starting index and length. SequenceManipulation creates a segment containing the first 25 symbols.

**Range** returns an **ISequence** interface to a **DerivedSequence** object. This object 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 change the underlying sequence. However, when you do so, the contents of the segment change to reflect the modified sequence.

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

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

These properties 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.

### [4] Modify the Sequence

The Sequence **object** supports a set of methods that manipulate the contents of the sequence in various ways. SequenceManipulation demonstrates several of these operations.

Change Symbols

You can change symbols by simply assigning a new **SequenceItem** object to the appropriate sequence element. SequenceManipulation is a DNA sequence, so the available **SequenceItem** objects are from the **DnaAlphabet** class, which can be accessed through **Alphabets.DNA**.

SequenceManipulation changes the first symbol to ‘A’. You must perform this action on the sequence itself; you cannot modify a derived segment. However, when you display the segment again, it reflects the modified sequence.

To change the values of a series of symbols with a single line of code, call **Sequence.ReplaceRange** and specify the starting index and a string containing the new symbols.

Insert a Symbol

To add a symbol to a sequence, call **Sequence.Insert** and specify the index of the insertion point and the **SequenceItem** object to be inserted. SequenceManipulation inserts ‘T’ at index 4, and displays the modified segment. ‘T’ becomes the symbol for index 4, and the rest of the sequence is advanced by one.

Insert a Series of Symbols

To add a series of symbols to the sequence with a single line of code, call **Sequence.InsertRange** and specify the insertion index and a string containing the symbols to be inserted. SequenceManipulation inserts “AAA” at index 4.

Remove Symbols

To remove a series of symbols from a sequence, call **Sequence.RemoveRange** and specify the starting index and the number of symbols to be removed. SequenceManipulation uses **RemoveRange** to remove the four symbols that were added to the sequence earlier. SequenceManipulation also modifies the first symbol to restore the sequence to its original value.

There are two related methods that remove a single symbol.

**Sequence.RemoveAt** removes the symbol at a specified index.

**Sequence.Remove** removes the first instance of a specified **SequenceItem** object.

For example, passing **Alphabets.DNA.T** to **Remove** removes the first instance of ‘T’ from the sequence.

# Example: How to Submit an MBF Web Services Request

This section introduces the basic features of the MBF 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 MBF 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 MBF.dll and MBF.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.

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 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 MBF;

using MBF.Web;

using MBF.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 MBF 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 MBF Namespaces

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

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

MBF 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 MBF.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 MBF 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. Table 10 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.

Table 10. 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 **MBF.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 MBF: 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, MBF 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 **FindParserByFile** 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 MBF **Registrable** attribute to the class.

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

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

Table 11 lists the registrable components, and the associated interface.

Table 11. 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, apply the **MBF.Registration.Registrable** attribute to the class, and set it to **true**. The following example shows how to make a custom parser registrable.

[Registrable(true)]

public class MyCustomParser : BasicSequenceParser

{

...

}

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

To have MBF load the component, put the component’s DLL in the MBF Add-Ins folder. Figure 4 shows a typical location for this folder.

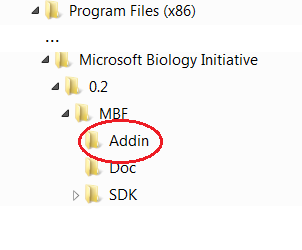


Figure 4. Add-ins folder

# Extending MBF: Data Virtualization

This section briefly describes the MBF data virtualization model, and is intended primarily for users who want to implement custom parsers that must load large numbers of data items from a file or large individual data items, such as a lengthy sequence. For detailed guidance, examine the code for a virtualization-enabled parser such as FastaParser, which is located in the source tree in the mbf\io folder.

## Data Virtualization Architecture

Figure 5 shows the data virtualization architecture as it is used by a parser.



Figure 5. Data virtualization architecture

The following is a brief description of each layer. The key components are described in the next section.

Applications

Applications typically use virtualization implicitly, by calling a virtualization-enabled parser and letting the parser decide whether to virtualize the data. Applications can also direct the parser to virtualize data by setting the parser’s **EnforceDataVirtualization** property.

Object Model

The object model acts as an intermediary between applications and stored data. Applications interact with object model in exactly the same way whether virtualization is enabled or not. If virtualization is enabled, the object model interacts with the virtual sequence provider to obtain the necessary data from the parser.

Virtual Sequence Provider

The virtual sequence provider is an abstraction layer between the object model and the parsers. The object model communicates with the provider’s upper edge, and lower edge communicates with the appropriate parser.

Parsers

MBF supports multiple parsers, each of which supports a particular format and storage type. The parser reads data from storage and converts the data from its native format to the MBF object model. Parsers must enable data virtualization if memory resources are insufficient, or if they are explicitly directed to do so by the application. If data virtualization is enabled, the parser interacts with the virtual list and then the virtual sequence provider to retrieve data from storage as needed.

**Important:** The maximum file size that can be handled by MBF data virtualization is 2 GB. Attempting to handle larger files can produce unreliable results.

Data Readers

Parsers typically use a data reader to simplify the process of retrieving data from storage. For example, parsers that read data stored as plain text files typically use a **MBF.IO.MBFTextReader** or **MBF.IO.MBFStreamReader** object to read the file line by line.

Storage

Data can be stored in a variety of ways. As a practical matter, it is usually stored as plain text files. However it could also be stored in a database such as SQL Server, in “cloud” storage such as Azure, and so on.

The optional “sidecar” file is created locally by some parsers to help manage data retrieval.

## How to Implement Data Virtualization

The following procedures describe the basic virtualization model for parsers. It assumes that the data is stored in a file, but the basics apply to any storage type.

To virtualize data

1. An application directs the parser to load a file that contains large amounts of data and the parser determines that the computer lacks sufficient memory resources to load the entire data set.

Alternatively, the application directs the parser to virtualize the data by setting **EnforceDataVirtualization**.

2. The parser creates a “sidecar” file and populates it with information to assist data retrieval.

The sidecar file is optional, but used by many parsers. The details are left to the parser developer, but the file usually contains basic information to assist the data retrieval process, such as file information, the number of sequences, a list of object file names, and so on.

3. The parser returns a **VirtualSequenceList** object that contains a list of the sequences in the data set.

4. The virtual sequence provider requests a sequence, specified by its index in the **VirtualSequenceList** object.

5. The parser gets a pointer to that sequence from the sidecar file and returns a **FileVirtualSequenceProvider** object for the sequence.

6. The virtual sequence provider calls the parser’s **ParseRange** method to request blocks of symbols.

7. The parser returns the requested block of symbols.

If the requested sequence is longer than 4Kb, the virtual sequence provider continues through the sequence 4Kb at a time, as the symbols are needed.

# Resources

This section provides links to additional information about MBF and related topics.

#### CodePlex Resources

Microsoft Biology Foundation

<http://mbf.codeplex.com>  
MBF\_Overview.docx  
MBF\_Programming\_Guide.docx  
MBF\_PaDeNA.docx  
MSR\_Sequence\_Assembler\_User\_Guide.docx

Research Biology Extension for Excel User’s Guide

<http://bioexcel.codeplex.com/>  
MBF\_Biology\_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/collaboration/tools/mbf.aspx>

Microsoft Research Biology Extension for Excel

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

#### 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>

NUnit integration for Visual Studio

<http://visualstudiogallery.msdn.microsoft.com/en-us/db5872a6-e5e5-4c1f-ad1c-a05cf2c143dc>

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

NUnit

<http://www.nunit.org/>

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 MBF RNA and protein alphabets.

## The RNA Alphabet

**RnaAlphabet** contains a set of fields, each of which contains an **MBF.Nucleotide** object that represents one of the members of the alphabet. Table B.1 lists the fields.

Table B.1. 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 MBF**.AminoAcid** object that represents one of the members of the alphabet. Table B.2 lists the fields.

Table B.2. 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 |