|  |  |
| --- | --- |
|  | Microsoft Biology Foundation Overview  Version 1.0 - June 2010 |

Abstract

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

This document gives an overview of MBF and one of its samples, BioDemo.py, an MBF demonstration written in the IronPython scripting language.

The Microsoft Biology Foundation is available at http://mbf.codeplex.com.

Contents

[Introduction 2](#_Toc265502857)

[Benefits of MBF 3](#_Toc265502858)

[How to Install MBF 3](#_Toc265502859)

[Prerequisites 3](#_Toc265502860)

[System Requirements 3](#_Toc265502861)

[Installation 4](#_Toc265502862)

[The MBF Architecture 4](#_Toc265502863)

[The MBF IronPython Samples 5](#_Toc265502864)

[The Library: BioIronPython.dl 5](#_Toc265502865)

[The Demo: BioDemo.py 6](#_Toc265502866)

[Solution Architecture 8](#_Toc265502867)

[Adding an IronPython Project to Visual Studio 8](#_Toc265502868)

[Running and Debugging the Code 9](#_Toc265502869)

[Resources 11](#_Toc265502870)

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

# Introduction

The Microsoft Biology Foundation (MBF) is an open source, reusable .NET Framework library and application programming interface (API) for bioinformatics research. MBF is designed to encourage extension, reuse, and community contribution via release as part of the CodePlex Open Source Initiative (OSI) at Microsoft.

Our primary goals are to enable participation by the bioinformatics community and to obtain a better technical understanding of the underlying object model, extensibility, and code architecture requirements to meet the needs of this community.

We encourage you to provide feedback on MBF at http://mbf.codeplex.com.

The bioinformatics community 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, GFF, GenBank, and BED.
* 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 applications can be implemented in a variety of .NET languages, including C#, F#, Visual Basic® .NET, and IronPython.

This document gives an overview of MBF and one of its samples, BioDemo.py, an MBF demonstration written in the IronPython scripting language. For information on how to develop MBF applications in other programming languages, see “Microsoft Biological Foundation Programming Guide” (MBF\_Programming\_Guide.docx in the MBF document folder).

You can also work with sequences using two tools included with MBF: Microsoft Research Biology Extension for Excel, an add-in for Microsoft Excel, and MBF Sequence Assembler, a .NET application. For more information, see these documents in the MBF document folder:

* Microsoft Biological Foundation Programming Guide   
  (MBF\_Programming\_Guide.docx)
* Microsoft Research Sequence Assembler: User Guide   
  (MSR\_Sequence\_Assembler\_User\_Guide.docx)
* Microsoft Research Biology Extension for Excel  
  (MBF\_Biology\_Extension\_User\_Guide.docx)

# Benefits of MBF

The bioinformatics field is still relatively young. The term ‘bioinformatics’ dates back to just the mid-1990s, and its practitioners come from a wide variety of backgrounds, particularly biology, physics, chemistry, and occasionally computer science.

Industry-standard architectural and coding practices are not consistently used by the bioinformatics community. Data formats are still poorly defined, with a number of emerging schemas capturing only some of the details. Scalability is becoming a serious problem as bioinformatics datasets grow in size. Simplistic implementations fail to execute on single CPUs and also have no provisions for leveraging multicore or multiprocessor architectures.

For these reasons, the need has been recognized for reusable libraries of high-quality bioinformatics code. The goal for MBF is to provide a development framework for the bioinformatics community that ensures the high architectural and coding standards necessary for ease of application extensibility and longevity.

# How to Install MBF

This section describes the prerequisites, system requirements, and installation steps for MBF.

## Prerequisites

To use the basic capabilities of MBF, you should have a basic understanding of:

* Methods and nomenclature of genomics and bioinformatics.
* Productivity software similar to Microsoft Office Excel.

To take advantage of programming and extensibility capabilities, you need at least:

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

## System Requirements

* Windows® XP Service Pack (SP) 2 and later versions of Windows.
* .NET Framework Version 4.0, available at <http://go.microsoft.com/fwlink/?LinkID=186913>

Additional software requirements for implementing MBF applications are described in “Microsoft Biology Foundation Programming Guide.”

## Installation

The installer for MBF is Setup\_Framework.msi, available at http://mbf.codeplex.com.

To install MBF

1. Copy the MBF installer to a folder on your hard drive.

2. Navigate to that folder and double-click Setup\_Framework.msi, which runs the installation wizard.

3. Follow the directions in the installation wizard to install MBF.

The MBF installer creates a directory named C:\Program Files (x86)\Microsoft Biology Initiative\1.0\MBF that contains the following:

\Add-ins

\MBF.PaDeNa.dll

\MBF.PAMSAM.dll

MBF.dll

MBF.WebServiceHandlers.dll

Readme.txt

If you install the optional SDK, the installer creates an SDK folder in the MBF folder that contains the following:

\SDK

\Samples

\IronPython

\ReadGenerator

\SAMUtils

\TridentWorkflows

MBF.chm

MBF\_Coding\_Conventions.docx

MBF\_Commenting\_Conventions.docx

MBF\_Contribution\_Documentation\_Template.docx

MBF\_Contribution\_Guide.docx

MBF\_Getting\_Started.docx

MBF\_Onboarding.docx

MBF\_Overview.docx

MBF\_PaDeNa.docx

MBF\_Programming\_Gudie.docx

# The MBF Architecture

The MBF contains libraries of reusable bioinformatics functions and algorithms using the .NET Framework, as shown in Figure 1. Each library has its own namespace and associated classes. See BioDotNet.chm in the SDK folder for a complete reference.

* **MBF:** An object model for storing sequence data, metadata, and encodings
* **MBF.Web**: Web services interface for connecting object model to various web-based components. BLAST and ClustalW are default implementations.
* **MBF.Algorithms**: Algorithms for translation, pair-wise and multi-sequence alignment, and sequence assembly.
* **MBF.IO**: Parsers and formatters for various forms of genomic data.



Figure 1. The MBF Architecture.

# The MBF IronPython Samples

IronPython is an open-source implementation of the Python programming language that is tightly integrated with the .NET Framework. IronPython can use the .NET Framework and Python libraries, and other .NET languages can use Python code just as easily. IronPython is available at http://ironpython.codeplex.com/.

MBF includes two IronPython samples:

* BioIronPython.dl  
  A library to simplify using Python as a scripting language for rapid development.
* BioDemo.py  
  A demonstration of some of the current non-GUI MBF features.

## The Library: BioIronPython.dl

BioIronPython.dl gives fast Python access to:

* Opening and saving sequence files of any type that MBF can parse, through the BioIronPython.IO module.
* Randomized sequence splitting, through the BioIronPython.Util module.
* Assembly, through the BioIronPython.Algorithms module.
* BLAST searches, through the BioIronPython.Web module.
* The C# MBF code directly, also through the BioIronPython.Util module.

## The Demo: BioDemo.py

In this section, we walk through the entire BioDemo.py script and describe each section of the code.

1. Import references for initialization.

# Copyright Microsoft Corporation. All rights reserved.

import clr

import sys

import time

# Adding the dll reference will throw an exception if we're debugging in VS

# from the Python development dir, instead of the standard non-dev method of running

# from the bin\Debug dir or an

# installation dir.

try:

clr.AddReferenceToFile("BioIronPython.dll")

except:

default\_filename = "bin\\Debug\\ATF14F8.gbk"

else:

default\_filename = "ATF14F8.gbk"

from BioIronPython.Algorithms import \*

from BioIronPython.IO import \*

from BioIronPython.Util import \*

from BioIronPython.Web import \*

again = "y"

2. Prompt the user for a sequence filename.

This can be any of the types of files parsed by MBF, but should contain at least some sequence data for the first sequence in the file.

print "Welcome to the Bio.NET IronPython Demo!"

while "yY".find(again[0]) != -1:

try:

# parse file

filename = raw\_input("\nPlease enter a sequence filename (defaults to " + default\_filename + "): ")

if filename == "":

filename = default\_filename

seq = open\_seq(filename)[0]

print "\nSuccessfully loaded sequence!"

print " ID = " + seq.ID

print " Length = " + `seq.Count` + "\n"

3. Load the first sequence from the file.

Display the ID and length of the sequence.

# create fragments

fragments = split\_sequence(seq.Range(0, 500), 10, 50)

print "A subsequence consisting of the first 500 nucleotides or amino acids has been split into",

print `len(fragments)` + " fragments, each of length 50."

print "These will now be reassembled! (This may take a minute.)\n"

4. Randomly break the sequence into multiple overlapping fragments of the same length, with sufficient coverage for reassembly (10x).

Display the number and length of the fragments.

Assemble the fragments into contigs, and sort the contigs in descending order by length.

Display the number of contigs formed and the length of the longest contig.

# assemble sequence and sort contigs by descending length

assembly = assemble\_pairwise(fragments)

contig\_list = sorted(assembly.Contigs, lambda c1, c2: c2.Length - c1.Length)

print "The fragments have been assembled into " + `len(contig\_list)` + " contigs, with",

print `len(assembly.UnmergedSequences)` + " unmerged fragments."

print "The longest contig has a length of " + `contig\_list[0].Length` + "."

print "Let's do a BLAST search with it. (This may also take a minute.)\n"

5. Run a BLAST search using the longest contig. Display the hits in a table.

# run BLAST search

job\_id = submit\_blast\_search(contig\_list[0].Consensus)

# wait for response

for i in range(1, 13):

time.sleep(5)

result\_list = poll\_blast\_results(job\_id)

if result\_list != None:

print "\nThe following results were returned:\n"

print "ID".ljust(40), "Accession".ljust(20), "Length".rjust(10)

print "-----------------------------------------------------------------"

for result in result\_list:

for record in result.Records:

for hit in record.Hits:

print hit.ID.ljust(40), hit.Accession.ljust(20), `hit.Length`.rjust(10)

print

break

6. If an error occurs at any point, display an error message and proceed to Step 7.

elif i % 2 == 0:

print "No response yet after " + `5\*i` + " seconds..."

else:

print "\nNo results have been returned from the BLAST search."

print "Giving up on job ID " + `job\_id` + "\n"

except:

print "An error occurred: " + `sys.exc\_info()` + "\n"

7. Ask if user would like to run the demo using another sequence.

# prompt to go again

again = " "

while "yYnN".find(again[0]) == -1:

again = raw\_input("Would you like to enter another sequence? (y/n): ")

if len(again) == 0:

again = " "

## Solution Architecture

We recommend that the IronPython code should be imported into the Visual Studio MBF solution. The code can then be modified and debugged easily with the MBF code it accesses. Visual Studio is the recommended Microsoft development environment for IronPython.

Visual Studio does not come with built-in support for IronPython, so there is no defined project type to contain, build, run, or debug Python files. An extension called IronPython Studio adds this basic functionality, but there are several reasons to avoid using the Python-friendly project types that IronPython Studio defines:

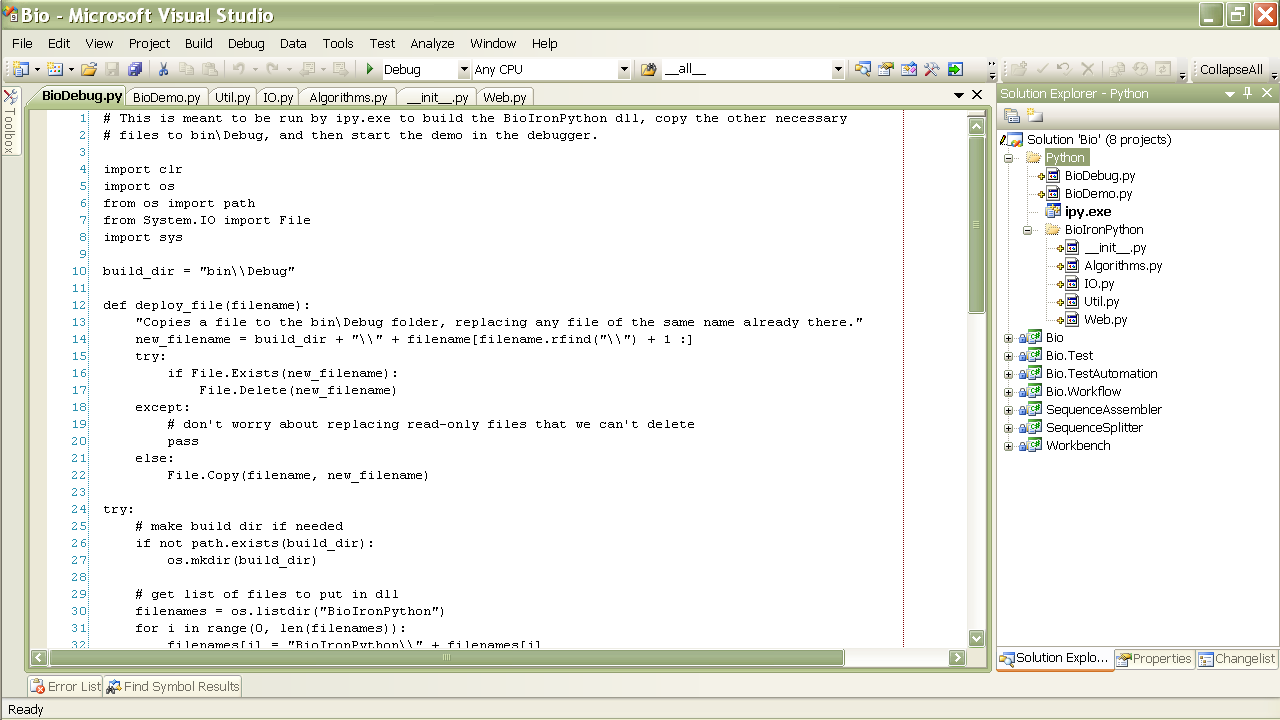
* Use of these project types would make it impossible to open the full MBF solution without first installing IronPython Studio.
* IronPython Studio is currently only compatible with IronPython 1.0. This means that many modules that Python developers often depend on would not be accessible.
* The DLLs that IronPython Studio builds do not work correctly.
* There are workarounds to be able to build, run, and debug Python files without using any of the built-in Visual Studio project types or adding any extensions.

## Adding an IronPython Project to Visual Studio

You can import your executable files into Visual Studio using the Add Existing Project command. Your IronPython application can then be debugged similar to a normal project. Right-click the executable icon to display a menu option to change the project’s properties, which include the execution target, working directory, and command-line arguments.

Using Visual Studio in this way enables the following solution:

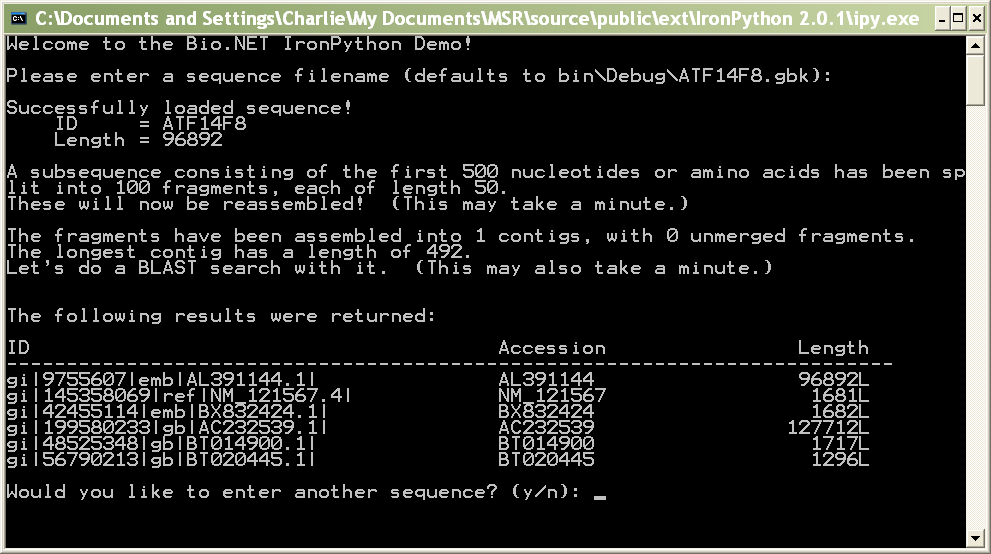
* The IronPython files reside in a folder at the same level as the C# projects, as shown in Figure 2.

  
Figure 2. A Python project imported into Visual Studio®

* The demo code is contained in Python\BioDemo.py, the library modules that comprise BioIronPython.dll are in Python\BioIronPython, and the build/debug script is BioDebug.py.
* The IronPython console executable, ipy.exe, is included alongside of the .py code files.
* In the ipy.exe properties, the working directory has been changed to the Python folder, and the command-line arguments set to -D BioDebug.py. The -D signifies the use of the debugger. The second argument is the file to be executed in the console.
* When Ipy.exe is set to be the startup project, BioDebug.py will be run through the Visual Studio debugger in the Python console.
* Running BioDebug.py builds BioIronPython.dll, copies all of the necessary files to the bin\Debug folder, and then starts BioDemo.py in the debugger, in the same way that a normal Visual Studio project is debugged.
* Developers who want syntax highlighting and other functionality for writing and debugging IronPython code can install IronPython Studio.

## Running and Debugging the Code

The demo can be debugged from within Visual Studio (or your IDE of choice), run from the IronPython console, or run from the command prompt. BioIronPython.dll can also be accessed directly from the IronPython console. The output will display as shown in Figure 3.

  
Figure 3. The IronPython output

To run the demo from the IronPython console

1. To execute the demo from the IronPython console, first copy the contents of Python\bin\Debug to your working directory, or switch your current directory to Python\bin\Debug.

2. Then run:

>>>import BioDemo

Note: Any commands at the global level of a Python file are executed when the file is imported.

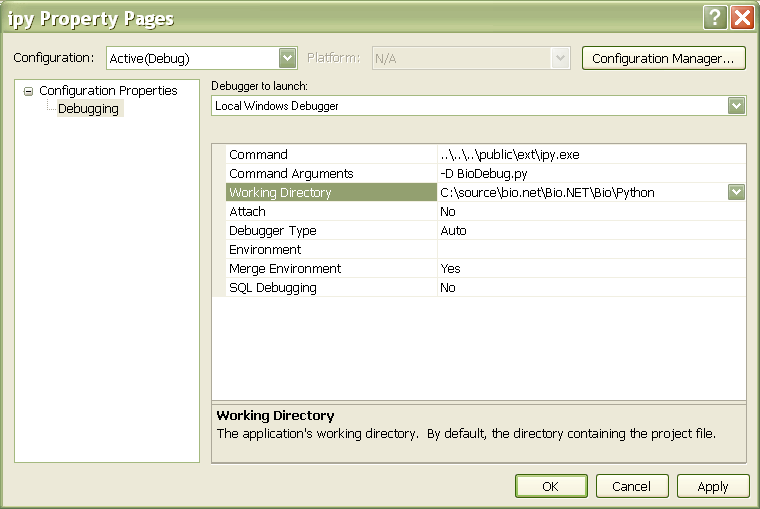
To run the demo from the command prompt

* Execute Ipy.exe, with the correct path to Python\bin\Debug\BioDemo.py as the only argument.

To debug the demo within Visual Studio

1. Right-click the Ipy.exe icon in the Solution Explorer and click Properties.

2. Set the properties as shown in Figure 3.

  
Figure 3. The properties of Ipy.exe

Note: The Working Directory needs to be an absolute path.

3. Set Ipy.exe as the startup project and press F5.

Put a breakpoint at the beginning of BioDemo.py if you want to step through it in the debugger.

Note: When debugging in Visual Studio, you might get an IronPython.Runtime.Exceptions.GeneratorExitException when BioDebug.py starts. Ignore it and press F5. The code will continue to run as usual.

To debug the demo outside Visual Studio

* If you haven’t built the code, do so by setting Ipy.exe as the startup project, and press F5.

If you don’t want the demo to run each time you build, comment out the line “import BioDemo” near the end of BioDebug.py.

# Resources

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

Microsoft Resources

IronPython

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

Microsoft Biology Foundation at Microsoft Research

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

Visual Studio 2010 and .NET Framework 4

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

CodePlex Resources

Microsoft Biology Framework

* http://mbf.codeplex.com  
  Microsoft Biology Foundation: Overview   
  Microsoft Biology Foundation: Programming Guide   
  Microsoft ResearchSequence Assembler:User Guide  
  PaDeNA: Parallel DeNovo Assembler

Research Biology Extension for Excel User’s Guide

<http://bioexcel.codeplex.com/>  
Microsoft Biology Extension for Excel: User Guide

Sandcastle

Sandcastle - Documentation Compiler for Managed Class Libraries  
<http://sandcastle.codeplex.com/>   
Sandcastle Help File Builder  
<http://www.codeplex.com/SHFB>

Bioinformatics References

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