|  |  |
| --- | --- |
|  | IronPython Programming Guide and Sample  Version 2.0.Beta - April 2011 |

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 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 “Programming Guide” ” at [CodePlex](http://mbf.codeplex.com/documentation) or the..\MBI\Doc folder.

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

Contents

[Introduction 2](#_Toc289768557)

[How to use the IronPython Samples 2](#_Toc289768558)

[The Library: Bio.IronPython.dl 3](#_Toc289768559)

[The Demo: BioDemo.py 3](#_Toc289768560)

[Solution Architecture 5](#_Toc289768561)

[Adding an IronPython Project to Visual Studio 6](#_Toc289768562)

[Running and Debugging the Code 9](#_Toc289768563)

[Resources 12](#_Toc289768564)

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 the project at <http://mbf.codeplex.com>.

MBF applications can be implemented in a variety of .NET languages, including C#, F#, Visual Basic® .NET, and IronPython. 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/>.

This document gives tips on how to use IronPython and provides an overview of one of its samples, BioDemo.py, which is an MBF demonstration written in the IronPython scripting language. For information on how to develop MBF applications in other programming languages, see the “Programming Guide” at [CodePlex](http://mbf.codeplex.com/documentation) or the..\Bio\Doc folder.

You can also work with sequences using two tools included in the project: Microsoft Research Biology Extension for Excel, an add-in for Microsoft Excel, and MBF Sequence Assembler, a .NET application.

For more information, see following documents at [CodePlex](http://mbf.codeplex.com/documentation) or the ..\Bio\Doc folder:

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

# How to use the 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/>.

The IronPython sample, BioDemo.py, is included in the project. It demonstrates some of the current non-GUI project features.

## The Library: Bio.IronPython.dl

BioIronPython.dl gives fast Python access to:

* Opening and saving sequence files of any supported type for parsers, 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# project 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

import os

from os import path

# 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("Bio.IronPython.dll")

except:

default\_filename = "bin\\Debug\\Small\_Size.gbk"

else:

default\_filename = "Small\_Size.gbk"

from BioIronPython.Algorithms import \*

from BioIronPython.IO import \*

from BioIronPython.Util import \*

from BioIronPython.Web import \*

build\_dir = "bin\\Debug"

def deploy\_file(filename):

"Copies a file to the bin\Debug folder, replacing any file of the same name already there."

new\_filename = build\_dir + "\\" + filename[filename.rfind("\\") + 1 :]

try:

if File.Exists(new\_filename):

File.Delete(new\_filename)

except:

# don't worry about replacing read-only files that we can't delete

pass

else:

File.Copy(filename, new\_filename)

try:

# make build dir if needed

if not path.exists(build\_dir):

os.mkdir(build\_dir)

# copy test file

deploy\_file("Data\\Small\_Size.gbk")

except:

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

raw\_input("Press enter to exit: ")

again = "y"

2. Prompt the user for a sequence filename.

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

print "Welcome to the Bio 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.

if seq.Count >= 500:

# 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\_string = poll\_blast\_results(job\_id)

if result\_string != None:

result\_list = parse\_blast\_results(result\_string)

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"

else:

print "Input sequence must have atleast 500 basepairs."

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 you import the IronPython code into the Visual Studio Bio.sln solution. The code can then be modified and debugged easily in conjunction with the MBF code it accesses. Visual Studio is the recommended Microsoft development environment for IronPython.

Visual Studio does not provide built-in support for IronPython. There is no defined project type to contain, build, run, or debug Python files. The IronPython Studio extension at <http://ironpythonstudio.codeplex.com/> adds this basic functionality, but the Python-friendly project types that IronPython Studio defines have shortcomings:

* Use of these project types would make it impossible to open the full MBF solution without first installing IronPython Studio.
* IronPython Studio currently is only integrated with Visual Studio® 2008 and only supports ironpython 1.0. This means that many modules that Python developers often depend on would not be accessible.

**Note**: Visual Studio® 2010 or later is required to build the Bio.sln solution.

* The DLLs that IronPython Studio builds do not work correctly.

In addition there are workarounds that enable you 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 a Visual Studio solution by using the Add Existing Project command. Your IronPython application can then be debugged similar to a normal project.

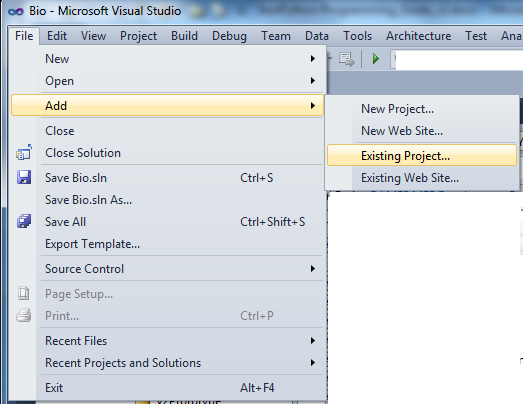
You can add an existing project to a solution and then edit that project to meet the requirements of the current solution.

To add an existing IronPython ipy.exe to a Visual Studio solution

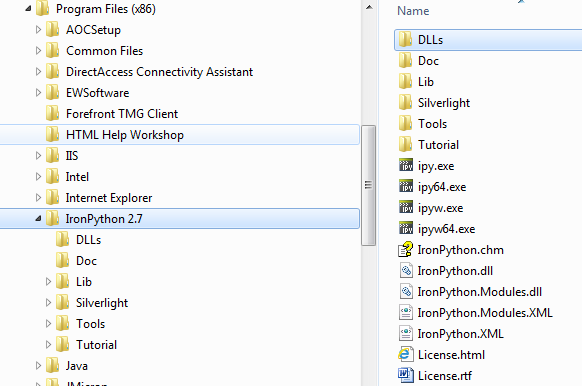
1. Open your Visual Studio solution.
2. In **Solution Explorer**, select the Bio.sln solution. Add the IronPython ipy.exe file to this solution.

**Note**: You must have previously downloaded IronPython from CodePlex, see [How to use the IronPython Samples](#_How_to_use).

1. On the **File** menu, point to **Add** and then click **Existing Project**.

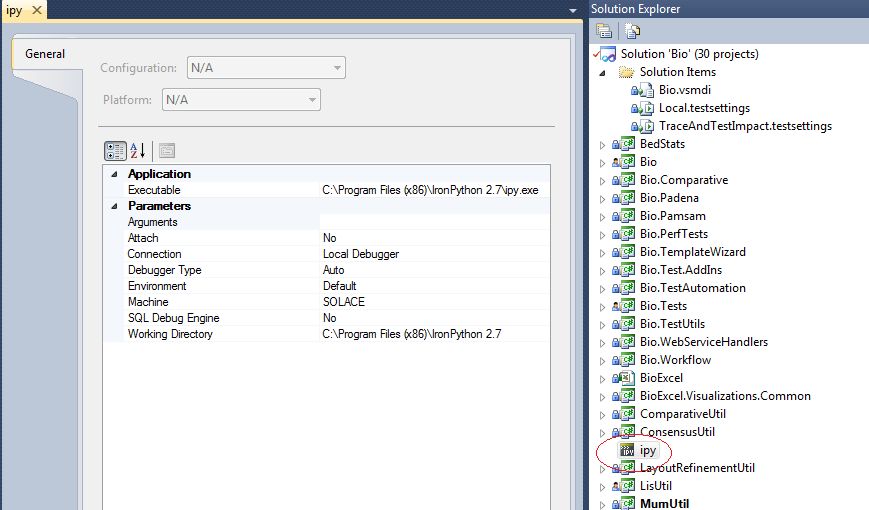


1. Navigate to the IronPython ipy.exe in the location where you installed IronPython, as illustrated in the following screen shot, and select it to add to it to the solution.



**Note**: The **Add/New Project** and **Add/Existing Project** commands can also be accessed by right-clicking the solution in Solution Explorer.

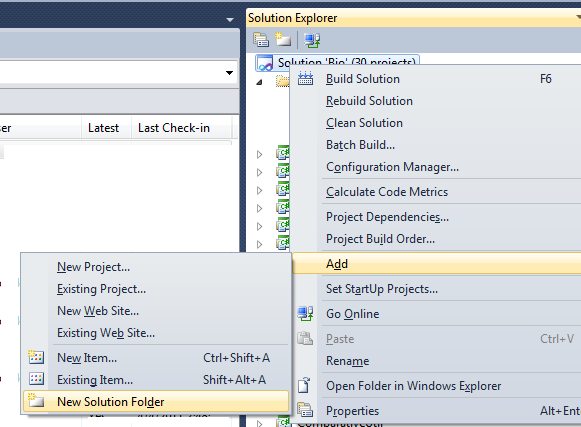
You can right-click the executable icon to display a menu option to change the project’s properties including the execution target, working directory, and command-line arguments.



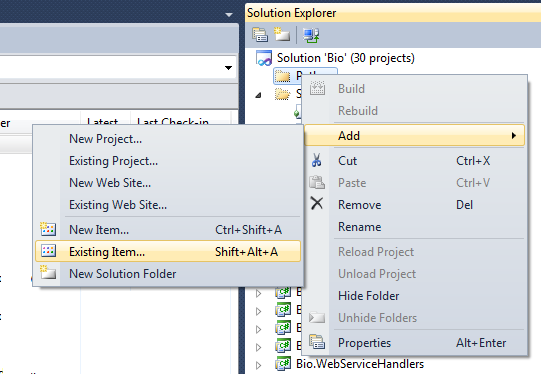
Now add the IronPython scripts shipped as part of MBF. Create 2 new solution folders in the Visual Studio solution and populate them with the IronPython files.

To add the existing Python project files and folders to a Visual Studio solution

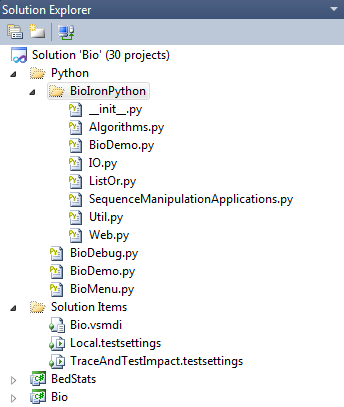
1. Right click the Bio.sln solution **Solution Items** folder, point to **Add** and select **New Solution Folder.** Name it “Python”.



1. Right click the **Python** folder, point to **Add** then click **Existing Item**.



1. Add the demo files by navigating to the Python demo files, in the ..\Source\Tools\Python folder, select them all and click the **Add** button.
2. Right click on the **Python** solution folder and add another solution folder. Name it “BioIronPython”.
3. Repeat step 2 for the **BioIronPython** folder. Add the demo scripts in the source ..\\Tools\Python\BioIronpython folder.
4. The new solution is illustrated by the following screen shot.

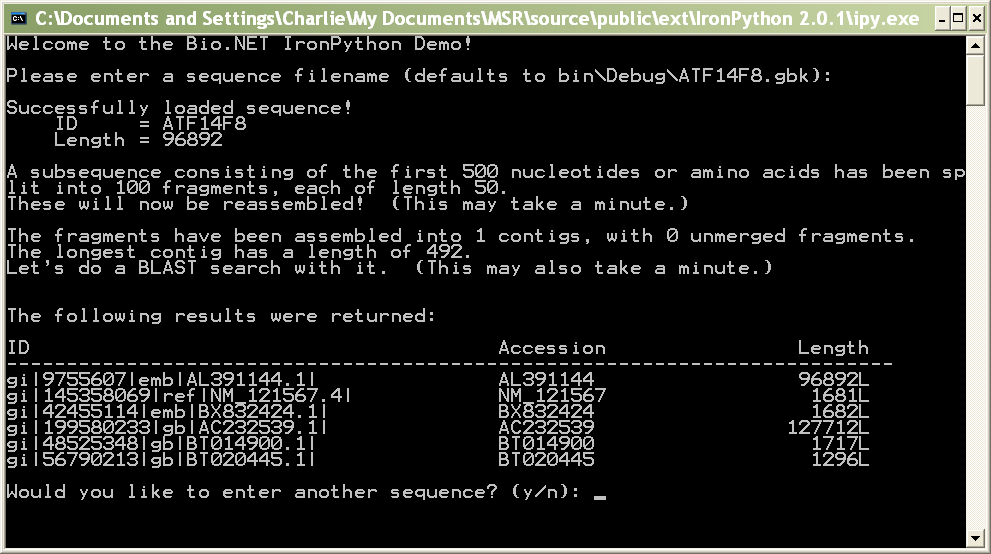


Your new solution will now have the following characteristics:

* The IronPython files reside in a folder at the same level as the C# projects.
* 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 the .py code files.
* In the ipy.exe properties, the working directory has been changed to the Python folder, and the 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 at <http://ironpythonstudio.codeplex.com/>

## 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 the following figure.

  
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.

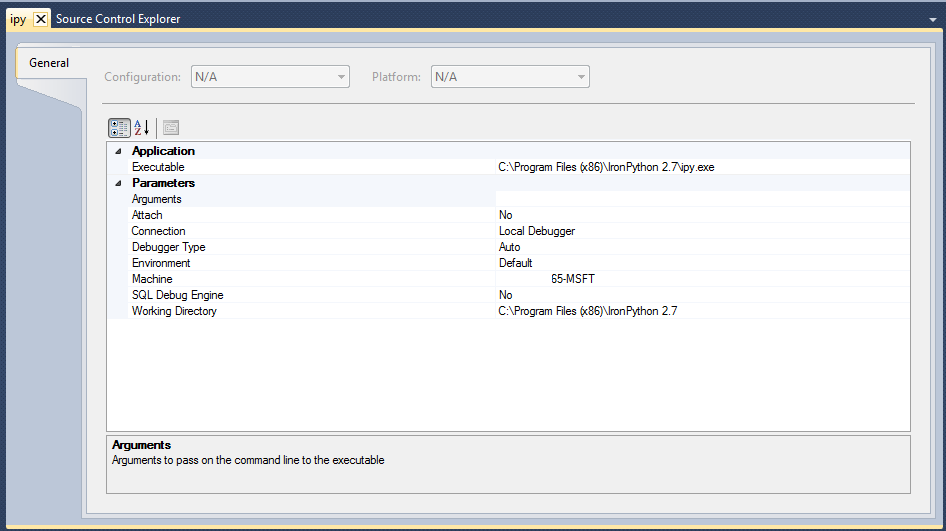
To run the demo from the command prompt

* 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 the following figure.

  
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 Beta 2

<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