**Bio.NET IronPython**

**README**

Contents

[1 Purpose 2](#_Toc233969717)

[1.1 The Library: BioIronPython.dl 2](#_Toc233969718)

[1.2 The Demo: BioDemo.py 2](#_Toc233969719)

[2 Solution Architecture 3](#_Toc233969720)

[2.1 The Wrong Way 3](#_Toc233969721)

[2.2 And Now for Something Completely Different 3](#_Toc233969722)

[3 Running the Code 4](#_Toc233969723)

[3.1 From within Visual Studio 2008 5](#_Toc233969724)

[3.2 From outside Visual Studio 5](#_Toc233969725)

[3.2.1 From the IronPython Console 5](#_Toc233969726)

[3.2.2 From the Command Prompt 5](#_Toc233969727)

[4 Extending the Code 5](#_Toc233969728)

[4.1 Python 6](#_Toc233969729)

[4.2 IronPython 6](#_Toc233969730)

[4.3 IronPython Studio 6](#_Toc233969731)

[4.4 Pitfalls 6](#_Toc233969732)

[5 There Is No Fifth Thing 7](#_Toc233969733)

# Purpose

The purpose of the Bio.NET IronPython code is to:

1. Provide a library to make it easier for programmers to use Python as a scripting language for rapid development of custom applications.
2. Provide a demo of some of the current non-GUI Bio.NET features.

## The Library: BioIronPython.dl

BioIronPython.dll gives fast, Python access to:

1. Opening and saving sequence files of any type that Bio.NET can parse, through the BioIronPython.IO module.
2. Randomized sequence splitting, through the BioIronPython.Util module.
3. Assembly, through the BioIronPython.Alborithms module.
4. BLAST searches, through the BioIronPython.Web module.
5. The C# Bio.NET code directly, also through the BioIronPython.Util module.

## The Demo: BioDemo.py

BioDemo.py does the following through a textual interface:

1. Prompts the user for a sequence filename.  This can be any of the types of files parsed by Bio.NET, but should contain at least some sequence data for the first sequence in the file.
2. Loads the first sequence from the file; displays the ID and length of the sequence.
3. Randomly breaks the sequence into multiple overlapping fragments of the same length, with sufficient coverage for reassembly (10x); displays the number and length of the fragments.
4. Assembles the fragments into contigs, and sorts the contigs in descending order by length; displays the number of contigs formed, and the length of the longest contig.
5. Runs a BLAST search using the longest contig; displays the hits in a table.
6. If an error occurs at any point, displays an error message and proceeds to step 7.
7. Asks if user would like to run the demo using another sequence.
   * If yes, returns to step 1.
   * If no, exits.

# Solution Architecture

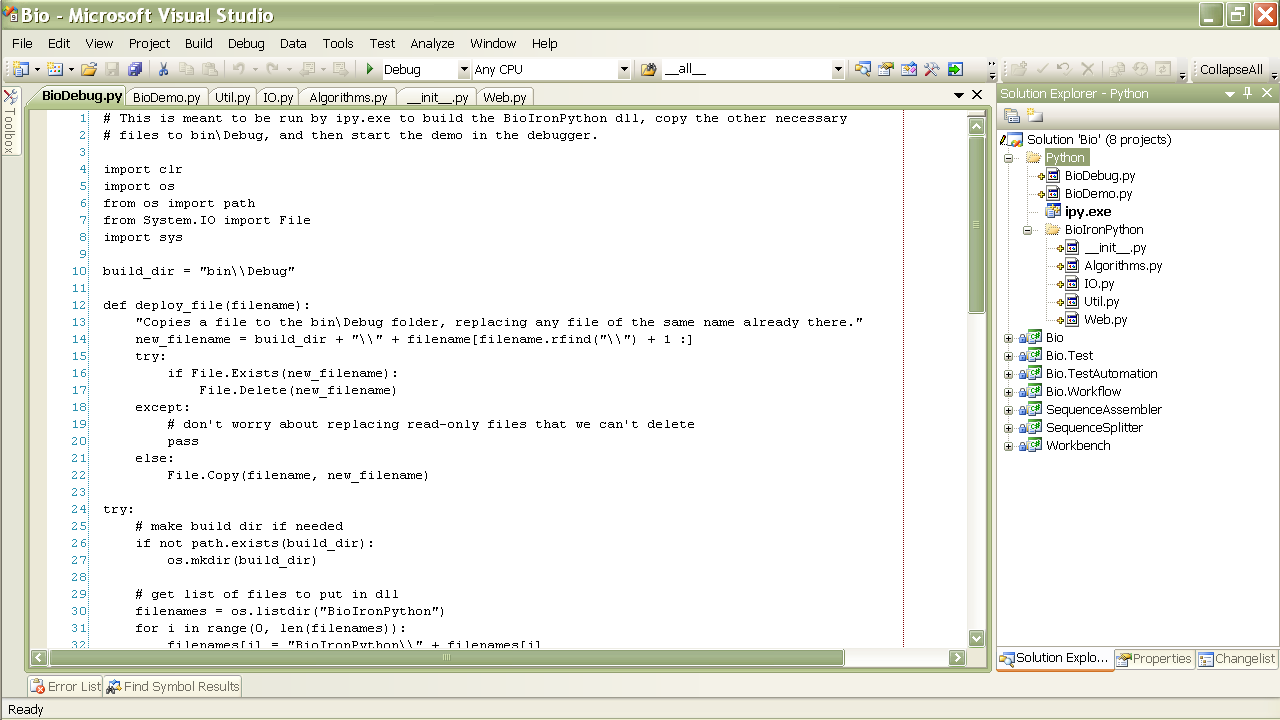
It is desirable for the IronPython code to be contained in the Visual Studio Bio solution, so that it can be modified and debugged fluidly with the Bio.NET code it accesses, and because Visual Studio is the preferred Microsoft development environment for IronPython.

## The Wrong Way

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

1. Use of these project types would make it impossible to open the full Bio solution without first installing IronPython Studio.
2. 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.
3. The dlls that IronPython Studio builds do not work correctly.
4. 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…

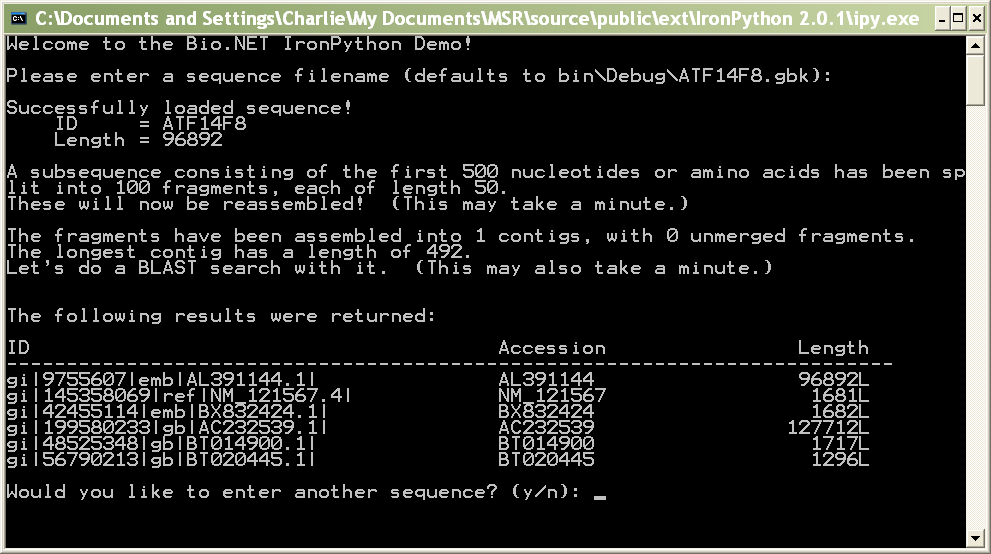
## And Now for Something Completely Different

Visual Studio allows executables to be imported using the “Add Existing Project” command, and then debugged similar to a normal project. Right clicking on the executable icon then brings up a menu option to change its properties, which include the execution target, working directory, and command-line arguments. This allows for the following solution:

1. The IronPython files reside in a folder at the same level as the C# Projects, as seen at right.

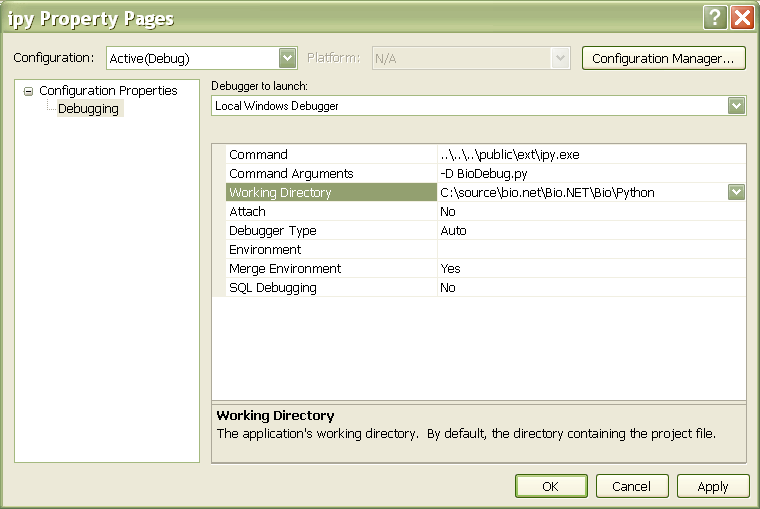
1. The demo code is contained in Python\BioDemo.py, the library modules that comprise BioIronPython.dll are in Python\BioIronPython, and the build/debug script is BioDebug.py.
2. The IronPython console executable, ipy.exe, is included alongside of the .py code files.
3. In the ipy.exe properties, the working directory has been changed to the Python folder, and the command-line arguments set to “-D BioDebug.py.” The –D signifies that we want to use the debugger. The second argument is the file to be executed in the console.
4. Now when ipy.exe is set to be the startup project, BioDebug.py will be run through the Visual Studio debugger in the Python console.
5. 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, just as would be expected when a normal Visual Studio project is debugged.
6. Developers who want syntax highlighting and some other bells and whistles for writing and debugging IronPython code can install IronPython Studio. (See [Section 4.3](#_Development_Environment).)

# Running the Code

The demo can be debugged from within Visual Studio (or your IDE of choice), or run from the IronPython console or the command prompt. BioIronPython.dll can also be accessed directly from the IronPython console. Wherever you run it from, the output will look something like this:

## From within Visual Studio 2008

To debug the demo within Visual Studio, you will first need to set the properties of ipy.exe. Right click on its icon in the Solution Explorer, and select Properties from the menu. They should match those seen below. The Working Directory needs to be an absolute path. The one used below is correct if you installed Bio.NET to C:\source\bio.net.



Once the properties are set, you just need to set ipy.exe as the startup project, and hit 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.  Just ignore it and hit F5. The code will continue to run as usual.[[1]](#footnote-1)

## From outside Visual Studio

If you haven’t built the code, do so by setting ipy.exe as the startup project, and hitting F5. If you don’t want the demo to run each time you build, comment out the line that says “import BioDemo” near the end of BioDebug.py.

If you haven’t installed IronPython on your machine, you can use the IronPython console executable included with the Bio.NET distribution. It is located at \public\ext\ipy.exe.

### From the IronPython Console

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. Then run

>>>import BioDemo

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

### From the Command Prompt

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

# Extending the Code

If you are going to extend the Bio.NET IronPython code base and are new to IronPython development, this is the place for you!

## Python

The version of IronPython that we use (2.0.1) is compatible with Python 2.5.2. The IronPython Tutorial assumes a prior knowledge of Python. There is a good Python tutorial at <http://www.python.org/doc/2.5.2/tut/tut.html>. Full appreciation of the examples in this tutorial—and every legitimate Python tutorial—requires a working knowledge of *Monty Python’s Flying Circus*, which can be acquired in your personal time at <http://www.youtube.com/MontyPython>.

## IronPython

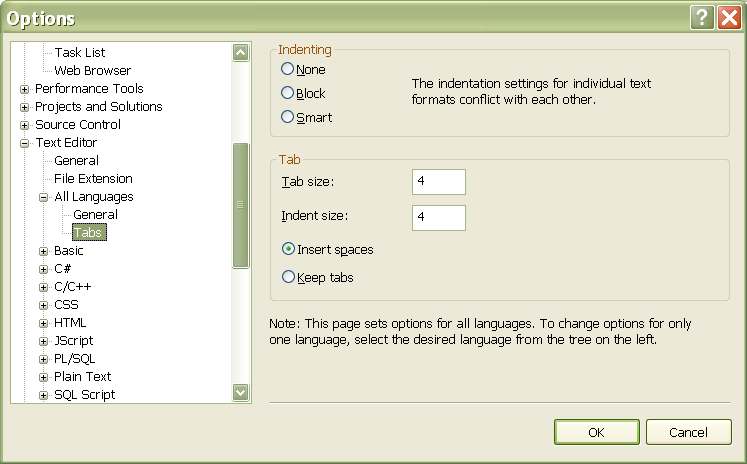
The full installation of IronPython can be downloaded from <http://ironpython.codeplex.com/Release/ProjectReleases.aspx?ReleaseId=12481#DownloadId=58403>. It includes a Tutorial and FAQ that will answer quite a few questions for new developers.

## IronPython Studio

You can develop IronPython code from Visual Studio 2008 without any additions, but there will be no syntax highlighting or code completion, and the debugger’s exception-handling functionality will be limited. These features can be added by installing IronPython Studio from <http://ironpythonstudio.codeplex.com/Release/ProjectReleases.aspx?ReleaseId=8934#DownloadId=30978>.

## Pitfalls

There are several difficulties to be aware of when using IronPython and IronPython Studio:

1. By default, using the tab key will insert actual tabs instead of spaces. IronPython does not acknowledge these as the same thing, and a mix of the two will cause runtime errors. Make sure your tab settings for “All Languages” match those pictured at right.
2. Python does not implicitly convert non-string types to strings during concatenation. Non-string data types should be converted explicitly by enclosing them in back ticks (`), which look unfortunately similar to apostrophes (').

my\_int = 5

print "The value of my\_int is " + `my\_int`

1. A Python list is not the same thing as a C# list. If you want to pass a Python list to a C# method, it will need to be copied to the appropriate C# type. Square brackets are used to denote generic types in Python.

c\_sharp\_seq\_list = List[ISequence](python\_seq\_list)

1. In Python, true and false are not keywords; use 1 and 0. Use not instead of an exclamation point to negate a Boolean value or expression, but use != to test if values are not equal. The Python equivalent of null is None.
2. The IronPython Studio project types cannot be opened by Visual Studio if the extension has not been installed. Do not include these project types in the Bio solution.
3. BioDebug.py cannot run the code that it has just put in the Python\bin\Debug folder, because it has already referenced the source files that define modules of the same names. It avoids name collisions by running the source files themselves. To run the code in the Python\bin\Debug folder through the Visual Studio debugger, target Python\bin\Debug\BioDemo.py by changing the command line arguments and the working directory in the Properties of ipy.exe.
4. Since the Visual Studio debugger is running the source files rather than the bin\Debug code, the path from the code to the location of the Bio.NET files copied to bin\Debug is different. This requires a workaround so that scripts that reference these assemblies can be run in either the Visual Studio debugger or the console. This is done through BioIronPython.Util.add\_biodotnet\_reference(dll\_name), which should be used whenever adding a reference to a Bio.NET assembly. When this method is called, it first tries to reference the assembly from the system path, which covers normal console use. If the dll is not found in the system path, the method then looks for a bin\Debug folder relative to the working directory. This allows the dll to found when the code is being run through the Visual Studio debugger.

# There Is No Fifth Thing



1. I tried my hardest to get this GeneratorExitException to go away, but to no avail.  It appears to be a bug in the IronPython type.py code, and I couldn’t find any useful hints online.  It only shows up when debugging, and it is fine to leave it unhandled; it just means you have to hit “return – return – F5” to ignore the error message every time you start up the demo in the debugger. [↑](#footnote-ref-1)