**MBF Self Registration**

[Overview 2](#_Toc242103005)

[Considerations 2](#_Toc242103006)

[Approach 2](#_Toc242103007)

[Scan 3](#_Toc242103008)

[Fetch 3](#_Toc242103009)

[Folder structure 3](#_Toc242103010)

[Design 3](#_Toc242103011)

[Interface Driven 3](#_Toc242103012)

[Control Flow 4](#_Toc242103013)

[Snippet 6](#_Toc242103014)

[Static and Dynamic collections 6](#_Toc242103015)

[Further Refinement 8](#_Toc242103016)

[Programming Primer 10](#_Toc242103017)

[Issues & Solutions 10](#_Toc242103018)

[Version 10](#_Toc242103019)

[Security 10](#_Toc242103020)

# Overview

This document provides the high-level architecture on self registration mechanism. This document is prepared with two purposes in mind. One, it is a proposal where MBF can be standalone module and newly implemented (by third party or open source community) modules can be recognized by MBF and the functionality will be available via MBF. Second, provide the necessary framework information to extend the MBF into the community development.

# Considerations

Before introducing the approach on self registration with MBF, the following points have been considered:

1. Maintaining the same code base with minimal change.
2. Also, there is newly extended module (PAMSAM) needs to be accommodated with minimal code change.
3. There is an assumption that community of Bio developers are not highly professional developers who can work with complex object model. So, the approach needs to be simplest and easy to use it.
4. Plug-n-play model.

# Approach

The approach is **Scan-n-Fetch**, where the plug-n-play is the base. MBF self registration module scans the specific folder (configurable) and fetches the specified interface driven objects and provides the instance to consumer application.

Here is the high-level interaction between MBF, consumer app, and third party module.

## Scan

MBF has object collection class (like, SequenceAligners, SequenceAssemblers, Alphabets and etc.,). Every collection classes call AssemblyResolver to specified interface driven objects.

As a first step in Scan, MBF scans itself for the specified interface driven objects (inside Bio.dll) and then scans specified folder with specified file or file types. MBF loads the file (third party implementation) using assembly reflection.

Folder scan allows configurable folder path and file filter type.

## Fetch

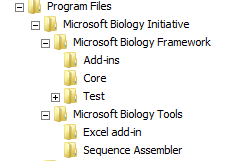
Once the file is loaded successfully, MBF exports all the types and look for specified interface type to be collected. Then it fetches the appropriate interface type using reflection and creates the instance of object.

Those newly created instances are added as part of object collection class and the consumer app can iterate thru instance collect as per the usage.

Fetch allows configurable Interface type.

## Folder structure

Here is the typical folder structure after the Installation of MBF.



The scan looks into Core and Add-ins folder under Microsoft Biology Framework.

# Design

## Interface Driven

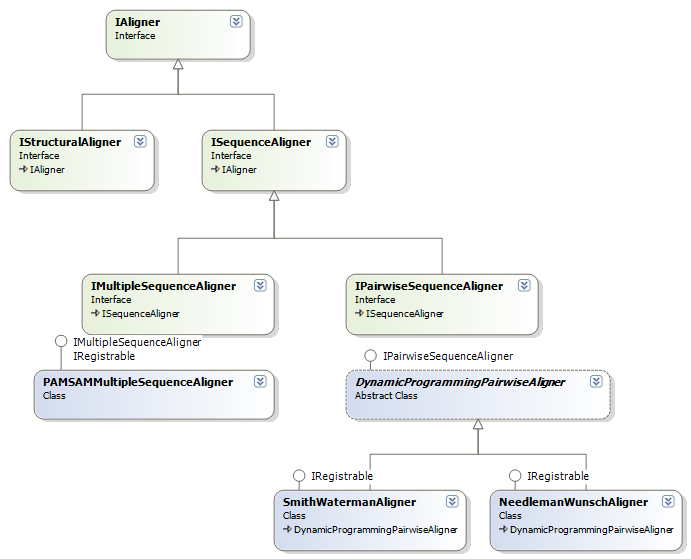
To make multiple implementations fit into MBF, it requires to use specified interfaces as defined by MBF. Also, the self registration gives the option to developer either it needs to be fetched by MBF or not. For that registration purpose, the implementation needs to follow MBF’s RegistrableAttribute attribute. When the class uses RegistrableAttribute, then it’s ready for auto registration.

The following class diagram shows the Interfaces which can be used in sequence alignment with the necessary implementation classes.

Since MBF already have the implementations of pair wise and multiple sequence alignment as separate modules, the common interface provides the missing connection between pair wise and multiple sequence alignment. This document focus on sequence alignment and structural aligner will be moved to feature development.

From the below class diagram, ISequenceAligner is the common interface which connects pair wise (IPairwiseSequenceAligner) and multiple (IMultipleSequenceAligner) sequence aligners. MBF already has the implementation of PAMSAMMultipleSequenceAligner, SmithWatermanAligner, NeedlemanWunschAligner and etc.,

Adding RegistrableAttribute as part of implementation class and driving from common interface is the only modification required.



## Control Flow

Once the necessary interfaces are used on implementation as per the guidelines, the interface fetch starts from object collection class (example, SequenceAligners). The fetch starts from the static constructor of object collection class. The constructor calls registration module twice, first time with the local reference and with the third party module specifics.

Integration of Bio.dll, Pamsam.dll and consumer app.

There are two utility classes used on registration, Registration and AssemblyResolver.

static SequenceAligners() calls Registration.RegistrableInstances()to all the instances of attribute object. This method intern looks for the objects which uses RegistrableAttribute using AssemblyResolver. AssemblyResolver directly involves with reflection to resolve the loaded assembly and exports the specified interface driven objects.

SequenceAligners provide the following output as defined in the above interface definition. The interface type will be either IAligner or ISequenceAligner. Since there are tree implementation uses RegistrableAttribute, the output will be the objects of:

1. SmithWatermanAligner
2. NeedlemanWunschAligner
3. PAMSAMMultipleSequenceAligner

### Snippet

Here is sample code in AssemblerResolver’s Resolve() method.

//output

List<object> resolvedTypes = new List<object>();

//use assebly reflection

Assembly assembly = Assembly.LoadFrom(filepath);

//get the types from reflection

Type[] types = assembly.GetExportedTypes();

//on each type

foreach (Type type in types)

{

……………

//check either its needed object or not

…………

//create the instace of the type

object obj = assembly.CreateInstance(types[i].FullName);

//add it as part of the output

resolvedTypes.Add(obj);

}

### Static and Dynamic collections

The object collection classes (here, SequenceAligners) can have static and dynamic collection of instaces. If the new implementation is clearly defined, then it can use static collections apart from dynamic collection. Dynamic collection is based on RegistrableAttribute. However, static collection is based on user defined.

Example, SequenceAligners has the following:

/// <summary>

/// A singleton instance of MUMmer class which implements

/// mummer alignment algorithm.

/// </summary>

private static MUMmer mummer3 = new MUMmer3();

/// <summary>

/// Gets an instance of MUMmer3 class which implements

/// the MUMmer algorithm for partial alignment

/// </summary>

public static MUMmer MUMmer

{

get

{

return mummer3;

}

}

/// <summary>

/// List of supported sequence aligners.

/// </summary>

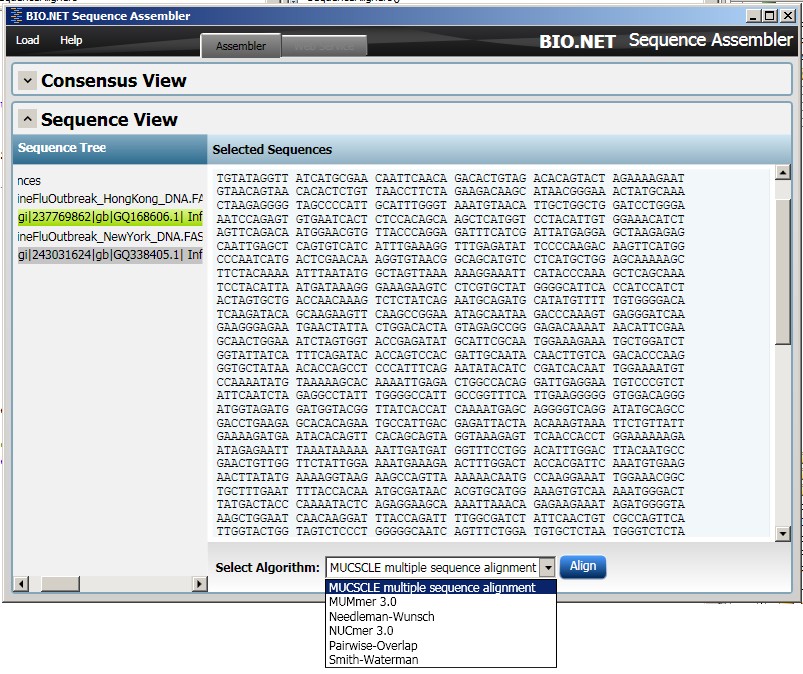
private static List<IAligner> all = new List<IAligner>()

{

SequenceAligners.mummer3

};

As per the above implemenation, MUMmer will be part of collection. However, MUMmer is not using RegistrableAttribute or registration mechanisam. The collection can have static and dynamic instances.



# Further Refinement

The above approach and design have been provided with the minimal code change on existing code base. This section is more focused on refactoring the existing object model and provide the safe registration.

On this proposal where MBF can hold only the core biology abstractions without any implementation. This core also allows basic implementation of core classes.

The core framework works on plug and play mechanism and any third party implements from the [interface guidelines](#1) will be available as part of object model.

MBF can be divided into two modules (DLLs).

1. MBF Core as just framework abstraction, interfaces and necessary base classes.
2. Default implementation of necessary functionalities provided by Microsoft to support MBF Core.

Then comes third party implementation, it is exactly similar to default Implementation. However, it is written by bioinformatics developer who wants to extend and introduce new functionalities like algorithms, parsers and etc. using MBF Core.

## Programming Primer

Here is code sample as mentioned in the above refinement section. The ZIP attached includes 4 different modules.

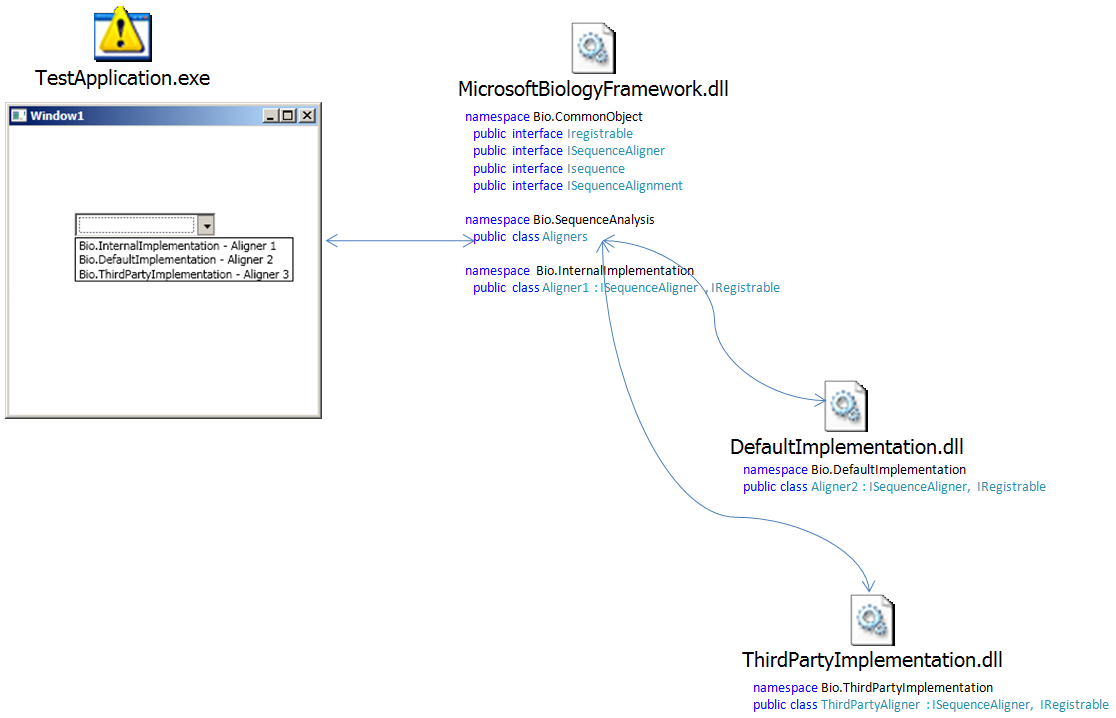


TestApplicaiton.exe – End user windows application which uses MBF.

MicrosoftBiologyFramework.dll – Microsoft’s Biology core framework.

DefaultImplementation.dll – Microsoft’s default implementation of core framework.

ThirdPartyImplementation.dll – Third party (open source) implementation of framework extensions.



# Issues & Solutions

## Version

Since MBF scans the specified folder, it controls the single version of same file name.

## Security

Since the entire MBF is part of open source community, the interface usage is the only controllable on any implementation.