|  |  |
| --- | --- |
|  | <Contribution Name> Programming and User Guide  .NET Bio Framework  Version 1.0 - June 2011 |

This document provides a template and some related guidance for documenting open source add-ins and new components for the .NET Bio Framework. See How to Use this Template.

Abstract

Your abstract should be a brief description of the document, preferably just a single paragraph. Its purpose is to provide readers with enough information to decide whether they want to read the document, no more. Put more detailed discussions in the “Introduction”.

This document describes …

<Put your author information here>

[.NET Bio Framework is an open source, reusable .NET library and application programming interface (API) for bioinformatics research software project provided under the Apache 2.0 license.](http://mbf.codeplex.com/license) This documentation 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.

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

Document History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Change |  |  |  |
| <1st pub date> | First publication | | | |
|  |  | | | |

Contents

[How to Use this Template [remove this section before publishing] 3](#_Toc256685141)

[Introduction 5](#_Toc256685142)

[Terminology 5](#_Toc256685143)

[Getting Started 5](#_Toc256685144)

[Prerequisites 5](#_Toc256685145)

[Hardware and Software Requirements 5](#_Toc256685146)

[Installation and Configuration 6](#_Toc256685147)

[How to Use <Contribution Name> 6](#_Toc256685148)

[<Tutorial 1> 7](#_Toc256685149)

[<Walkthrough Example 1> 7](#_Toc256685150)

[<Walkthrough Example 2> 7](#_Toc256685151)

[<Tutorial 2> 8](#_Toc256685152)

[Debugging Tips 8](#_Toc256685153)

[Troubleshooting Tips 8](#_Toc256685154)

[Resources 8](#_Toc256685155)

[Appendix A: <Title> 8](#_Toc256685156)

[Appendix B: <Contribution> API Reference 9](#_Toc256685157)

[Introduction to <API Name> 9](#_Toc256685158)

[<CLASSNAME> Class 9](#_Toc256685159)

[<DELEGATENAME> Delegate 12](#_Toc256685160)

[<ENUMNAME> Enumeration 12](#_Toc256685161)

[<ATTRIBUTENAME> Attribute 13](#_Toc256685162)

[<STRUCTNAME> Structure 13](#_Toc256685163)

# How to Use this Template [remove this section before publishing]

|  |
| --- |
| This document provides a template—following this section—that provides the basic structure for a User/Programming Guide for .NET Bio Framework contributions.  WHEN TO USE THIS TEMPLATE: This template should be used when contributing new functionality or features, or when making major changes to the framework. When making bug fixes, adding new components to existing functionality, or other changes where documentation already exists, please make sure to update the existing documentation, using this template only for guidance.  Note: The format is entirely flexible. You are free to adapt this template to the particular needs of your contribution.  To use the template: Replicate the sections you want to use, and add appropriate content, starting with “Introduction.” Delete this section.  Some basic suggestions:   * A Guide is an introduction, not an encyclopedia.   A Guide should give users a good head start on using the various parts of your contribution, but it should not attempt to cover every detail. Focus on the 80% stuff, not the corner cases.   * A Programming Guide is a practical How-To document.   Such as guide typically consists of tutorials, procedures, and so on, that explain in practical terms how to use your Framework contribution in an application.   * Your discussion of any API in the Guide should be representative, not exhaustive.   Guides should be accompanied by an API Reference, which contains all the details of how to use each API element. However, consider including a brief summary of the key parts of the API as an appendix.   * Consider writing or modifying samples specifically for the Guide.   This will ensure that samples are directly related to the topics you want to address, and that are written to provide straightforward and readable example code.   * Samples should be readable.   Eliminate any code that is not essential and relevant to the point you are trying to get across, such as routine error-correction or parameter validation code.  Conventions in this template   * <...> represents placeholder text that you should replace with appropriate content. * Text with the Instruction style (shaded, 9 pt) is explanatory and should be deleted from your actual Guide. * Elements in the template use Word styles from the Framework documentation set.   Our goal for providing this template is to help integrate Framework community contributions, so that contributions fit with the rest of the Framework documentation.   * The Table of Contents shows Heading 1 and Heading 2 headings.   To update the Table of Contents, place your cursor in the table and press F9. If the TOC is overly long, you can use the Table of Contents button on the Reference ribbon to reduce the maximum heading level to Heading 1. |

You can copy and use these components as needed in your documentation.

Table Head

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Normal |  |  |  |  |
| Normal | * Table Bullet |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Table #. <Name> Methods

|  |  |
| --- | --- |
| Method | Description |
| <Method> | Text description |
|  |  |
|  |  |

Table #. <Name> Parameters

|  |  |
| --- | --- |
| Key | Value |
| <Parameter> | Description |
|  |  |
|  |  |

Start of template

# Introduction

The Introduction provides a basic overview of your contribution and any API included. Provide readers with enough that they can understand the rest of the paper, but resist the temptation to go off on interesting digressions. Try to keep the introduction to no more than 10-20% of the total page count. If you want to go into detail about issues beyond “how to use <your contribution>,” consider writing a separate white paper.

Use subheads as needed to provide structure, but try not to go beyond Heading 3.

<Introductory text>

# Terminology

If your contribution uses a significant amount of specialized terminology, consider including a glossary of terms that are used in the Guide.

<Optional – If you choose to define special terms>

This section contains terminology that is specific to <Contribution Name>. It provides a reference for terms that are used later in this paper.

<Term1>

<Definition of Term1>

<Term2>

<Definition of Term2>

# Getting Started

Getting Started should have everything that users need to know or do before they start into the actual How-To part of the document. Use as many of the following sections as make sense for your contribution.

<Brief description of the contents of this section>

## Prerequisites

Describe anything that users should be familiar with before they start the How-To sections, including:

APIs (for example, be familiar with the WCF APIs).

Tools (for example, know how to use WinDbg to debug user-mode processes).

Languages (for example, be familiar with C#).

Basic technologies (for example, programming for Web services).

<Describe prerequisites.>

## Hardware and Software Requirements

Hardware. You must have the following hardware to implement Framework applications:

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

Describe any additional minimum hardware requirements and, optionally, the ideal hardware, including:

Computers (for example, requires at least a dual-core x86 computer with a clock speed of at least 2.5 GHz).

Video cards (for example, requires a DirectX 9-compatible video card).

Disk space (for example, requires at least 25 GB of available disk space).

Network cards (for example, needs at least 100 Mbit Ethernet, with Gigabit Ethernet preferred).

Other (RAM, specialty cards, ...)

Software. You must have the following software to implement <Contribution Name>:

* Windows® XP SP2 or later, x86 or x64 versions
* Visual Studio 2010, Beta 2 or later
* .NET Framework 4.0 Beta 2 or later

Important: To compile a project that uses .NET Framework 4.0 CLR, you must have Visual Studio 2010.

* Microsoft Silverlight 4 SDK
* Microsoft HPC Pack 2008 R2 Client Utilities Redistributable Package with Service Pack 1 (<http://www.microsoft.com/downloads/en/details.aspx?FamilyID=0a7ba619-fe0e-4e71-82c8-ab4f19c149ad>)
* .NET Bio 1.0 or later
* You can install the DLLs or build them yourself from the Framework source code, depending on which installation option you choose.

<Describe additional software requirements.>

Optional software includes:

<Optional - Describe any additional optional software requirements.>

Including:

Development tools

.NET version

SQL or other databases.

Others (WinDbg, SDKs, PERL, ...).

## Installation and Configuration

<Describe how to install and configure your contribution.>

Describe how to install and configure your contribution and the user’s system, so that it is ready to use for the How-To sections, including how to configure access settings correctly.

Provide a list of the important files and installation folders.

How to configure your contribution (or at least a basic default configuration to get users started).

Access configuration. Be thorough; debugging access issues tends to be very frustrating.

Configuration for related software, as needed (for example, must any of the related software run as Administrator?).

How to verify the installation.   
A simple verification program to test things like access rights can be a big help for this step.

## How to Use <Contribution Name>

<Describe how to use your contribution.>

For .NET Bio Framework contributions that expose a UI, or where the user might want to tweak the configuration in various ways:

How to change the configuration.

A user’s guide that describes how to use the UI.

If this requires more than a couple of pages, consider writing a separate user’s guide or integrate the information into the UI.

# <Tutorial 1>

<Optional – these are only suggestions for structuring your content if you choose to include a tutorial.>

Replace the heading with a descriptive title.

The first tutorial is typically a very simple “Hello World” sort of application, to walk the user through the essential steps for implementing solutions that use your contribution. Keep the code as simple as possible.

<Application description.>

Describe briefly what the solution does, from a user perspective. For solutions that have a UI, a screenshot or two can be helpful.

There are two basic ways to approach samples:

Walk the user through a completed sample.

Start with an incomplete example and have the user add code.

Take the walkthrough approach first

## <Walkthrough Example 1>

<Describe what the walkthrough will show, in reference to the Code Example.>

Program Listing Title

<Complete Code Example 1>

Put the sample code after the solution description. For a “Hello World” level of solution, you might be able to put all the code in a single block.

<Example 1 walkthrough.>

Take the user through the sample, one step at a time, explaining what is happening at the key points.

< Step 1 of Code Example 2>

If you have broken the sample code into multiple chunks, each chunk should be followed by a step-by-step description.

For longer samples, consider breaking the sample into manageable chunks. Alternatively, provide the sample separately and just put excerpts in this document.

## <Walkthrough Example 2>

<Describe what the walkthrough will show, in reference to the Code Example 2.>

The starting point is your first example:

This could be just a blank Framework application. For more complex samples, provide a separate sample application that the user can add code to.

<Describe how to create the starting point.>

Program Listing Title

<Code example 2 Starting Point>

Show the completed code and highlight the parts that the user must add. Include enough of the starting code to allow users to locate themselves in the solution.

<Example 2 walkthrough>

Walk users through the code that they will add to the starting solution, and describe how it works.

Program Listing Title

<Code example 2>

If the user needs to add more than a few lines of code, consider using multiple code examples, each followed by a walkthrough.

To compile and run the <sample name> example:

1. <Initial step (typically how to load the starting application)>

2. <...>

Provide a detailed description of how to compile and run the sample, including things such as:

Header files that must be included.

Assemblies that must be referenced.

How to create and format data files or how to access other types of data, such as data stored in a database.   
If this is a complicated procedure, put it a separate section, following the last walkthrough.

How to build the application, including any special compiler settings.

# <Tutorial 2>

<Optional – these are suggestions for structuring your content if you choose to include a tutorial.>

Replace the heading with a descriptive title

If your contribution has multiple capabilities that can’t be shown with one simple walkthrough, create additional tutorials.

# Debugging Tips

Provide advice on how to debug a solution that uses your Framework contribution, especially if the debugging involves some non-standard or sophisticated techniques. If you feel adventurous, create a bug in a sample application, and then walk them through how to track it down.

<Optional - if you choose to include Debugging advice.>

# Troubleshooting Tips

Provide advice on how to solve problems that users might encounter.

<Optional - if you choose to include Troubleshooting advice.>

# Resources

This section provides links to additional information about <Contribution Name> and related topics.

<Resource1>

<Link to Resource1>

<Resource2>

<Link to Resource2>

# Appendix A: <Title>

<Optional - if you choose to include appendix information.>

<Description>

Include appendices as appropriate. Use appendices for lengthy but useful text that doesn’t fit in the body of the document, especially if the information is relevant throughout the document. Examples include:

Lengthy tables, such as a complete list of configuration settings or op codes.

Details of data formats.

XML schemas.

# Appendix B: <Contribution> API Reference

We recommend creating your API reference using SandCastle with a Help file output. However, this appendix provides brief tips and basic formatting for creating an API reference.

<Optional - if you choose to include an API reference or summary.>

## Introduction to <API Name>

Include a brief introduction to the API and its use, plus a description of any documentation conventions that aren’t obvious.

Introductory text.

## <CLASSNAME> Class

A one-sentence description of the class. Put additional comments in Remarks.

Class description.

public class <NAME> : <ParentClassName>, <InterfaceName1>...

#### Types

The Types section is used only for generic classes. Delete the section for other classes.  
Delete the Types section if not needed.

T

Type description.

#### Remarks

Remarks, as appropriate.

#### Members

Put member names in the table in the same order as they are in the reference proper, and hyperlink each name to the associated topic heading later in the document. For overloaded methods, link only the first overload.

|  |  |  |
| --- | --- | --- |
| Method1 | ... |  |
| Method2 |  |  |
| ... |  |  |
| Property1 |  |  |
| Property2 |  |  |

### <CLASSNAME> Constructor (Type1, Type2, ...)

If possible, put (Type1, Type2, ...) on the same line as the name. If the list is too long for a single line, break as shown. Notice that the lines are broken by a soft line break (SHIFT-ENTER).

The purpose of the type list in the method heading is to distinguish overloads from each other. If the class has only one constructor, omit (Type1, Type2, ...) from the heading.

Add the standard boilerplate description for constructors. Just replace <NAME> with the class name.

Initializes a new instance of the <NAME> class.

public <CLASSNAME>(

Type1 param1,

Type2 param2,

...

)

Describe each parameter, how developers should use it, and any restrictions or requirements on what can be passed.

#### Parameters

param1

Description.

param2

Description.

#### Remarks

Remarks, as appropriate.

### ~<CLASSNAME> Destructor ( )

Omit the destructor topic, if the class does not have one.

Add the standard boilerplate description for destructors. Replace <CLASSNAME> with the class name.

The <CLASSNAME> destructor.

public ~<CLASSNAME>( )

### <METHODNAME> Method (Type1, Type2, ...)

A one-sentence description of the method. Put additional comments in Remarks.

Description.

public <ReturnType> <METHODNAME> (

<Type1> <param1>,

<Type2> <param2>,

...)

The Types section is used only for generic methods. Delete the section for other classes.

#### Types

T

Description.

Describe each parameter, how developers should use it, and any restrictions or requirements on what can be passed.

Prefix the description with [IN] [OUT] or [IN/OUT] if it isn’t apparent.

#### Parameters

<param1>

Description and usage.

<param2>

Description and usage.

#### Return Value

Describe the return value(s). For multiple return values, use a list or table. For void methods, omit the section.

Return value description

#### Exceptions

List the exceptions that the method throws, especially any custom exceptions. This section is optional, but sometimes very helpful to users.

Exception list.

#### Remarks

Remarks, as appropriate.

The Back to <NAME> Class line is a hyperlink back to the beginning of the class. It’s not strictly necessary, but is convenient for users.

### <PROPERTYNAME> Property

A one-sentence description of the property. Put additional comments in Remarks.

Description

Include set; and/or get; in the syntax block, to indicate whether the property is read/write, read-only, or write-only.

public Type <PROPERTYNAME>{set; get;}

Describe the property value, if it isn’t obvious from the description. If it is, you can omit this section. For enumerable values, use a list or table.

#### Property Value

Description.

#### Remarks

Remarks, as appropriate.

### <EVENTNAME> Event

A one sentence description of the event. Put additional comments in Remarks.

Description

Event handlers don’t necessarily incorporate the event name; use the appropriate handler

public event <EventHandler> <EVENTNAME>

#### Remarks

Remarks, as appropriate.

Fields are used infrequently, and are typically protected rather than public.

### <FIELDNAME> Field

A one sentence description of the field. Put additional comments in Remarks.

Description

protected <Type> <FIELDNAME>

Describe the field value, if it isn’t obvious from the description. If it is, you can omit this section. For enumerable values, use a list or table.

#### Field Value

Description.

#### Remarks

Remarks, as appropriate.

Delegates are H1 headings, and documented after the class references.

## <DELEGATENAME> Delegate

A one sentence description of the delegate. Put additional comments in Remarks.

Description

Event handlers and event argument objects don’t necessarily incorporate the delegate name, so use the appropriate names.

public delegate void <EventHandler>(

Object sender,

<EventArgs> e

)

#### Parameters

sender

Description

e

Description.

#### Remarks

Remarks, as appropriate.

## <ENUMNAME> Enumeration

Enumerations are H2 headings, and documented after the class references. If enumerations are used for only one method or property, they don’t need a separate reference topic. Just put the table in the appropriate method or property section (usually the Return Value section, or one of the parameter descriptions).

Add a one-sentence description of the enumeration. Put additional comments in Remarks.

Description.

public enum <ENUMNAME>

Note: The Value column is optional but sometimes useful, especially if the members represent separate bits.

#### Members

|  |  |  |
| --- | --- | --- |
| Name | Value | Description |
|  |  |  |
|  |  |  |
|  |  |  |

#### Remarks

Remarks, as appropriate.

Attributes are H1 headings, and documented after the class references. Most of the usage information for the attribute should go here. However, some usage information might fit better in the associated class reference.

## <ATTRIBUTENAME> Attribute

A one-sentence description of the attribute. Put additional comments in Remarks.

Description

[<ATTRIBUTENAME> (<named1>, <named2>,..., <positional1>, <positional2>, ...)]

Omit either or both Parameters sections, if they don’t apply to the attribute.

#### Named Parameters

<named1>

Description

<named2>

Description

#### Positional Parameters

<positional1>

Description

<positional2>

Description

#### Remarks

Remarks, as appropriate.

Note: Structures are H1 headings, and documented after the class references. This topic type is normally used only for C-style structures. Managed structures are typically documented like classes, as are C++ structures that have class-like features such as constructors or methods.

## <STRUCTNAME> Structure

A one sentence description of the class. Put additional comments in Remarks.

Description.

typedef struct \_<STRUCTNAME> {

Type1 member1;

Type2 member2;

...

} LIST\_ENTRY;

typedef <STRUCTNAME>, \*P<STRUCTNAME>;

List the members in the order that they are declared.

#### Members

member1

Description.

member2

Description.

#### Remarks

Remarks, as appropriate.

Standard C++ functions—i.e., exported by name and not part of a class—are H1 headings. They can appear before or after the class references, depending on their importance.