dotfuscator

Dotfuscator Software Services Community Edition Users' Guide

Version 5.0

PreEmptive Solutions

Table of Contents

Dotfuscator Software Services CE

Dotfuscator is the leading .NET obfuscator and compactor that helps protect programs against reverse engineering while making them smaller and more efficient. In addition, Dotfuscator now provides a way of injecting additional pre-built functionality that provides usage tracking, tamper detection, and expiration into your .NET applications.

In the Dotfuscator Software Services CE Documentation

Introduction

Explains the benefits of using Dotfuscator Software Services Community Edition 5.

Getting Started

Explains how to launch Dotfuscator Software Services Community Edition 5 and register your product.

Configuration Options

Explains how to obfuscate your application using renaming. Then explains how to add feature usage tracking, tamper detection, and expiration to your application using Dotfuscator's code injection capabilities.

Results

Shows a visual representation of your application after Dotfuscator rebuilds it.

Beyond Obfuscation

Explains the new category of detective control in Dotfuscator CE, allowing you to better monitor, manage, and protect your applications

Free Runtime Intelligence Services Portal

Describes the freely available reports and dashboards that give you insight into how your application is being used.

Introduction

Your copy of Microsoft Visual Studio 2010 includes a free license for PreEmptive Solutions' *Dotfuscator Software Services Community Edition 5* (Dotfuscator CE 5). Like earlier versions of Dotfuscator CE included in Visual Studio 2008, 2005 and 2003, it provides you with tools to protect and harden your .NET applications. Dotfuscator CE 5 works on compiled assemblies without the need for additional programming or even access to source code.

In addition to simple obfuscation, Dotfuscator CE 5 offers a range of new software services for developers, architects and testers. Examples of the new code protection, monitoring and management capabilities included in Dotfuscator CE 5 are:

- *Tamper defense* to detect the execution of tampered applications, transmit incident alerts, and terminate tampered sessions.
- *Application expiration* behaviors that encode an "end-of-life" date, transmit alerts when applications are executed after their expiration date, and/or terminate expired application sessions.
- *Session tracking* to determine what applications have been executed, what versions of those applications, and for how long.
- *Feature usage tracking* to determine what features are being used, in what sequence, and for how long.

In this section

Capabilities

This section focuses on the capabilities of Dotfuscator CE 5 with some references to advanced options available through upgrades.

Dotfuscator CE 5 is a *post-build* system for .NET applications. With Dotfuscator CE 5, Visual Studio users are able to obfuscate assemblies and inject tamper detection, application expiration, session monitoring, and feature tracking functionality – all without programming or access to the original source code. This information can be directed to one or more arbitrary endpoints (or clouds) for reporting and analysis.

Application Protection

Intellectual Property theft, piracy, and tampering each materially increase financial, operational, and reputational risk – for both application suppliers and consumers. Dotfuscator CE 5 includes baseline obfuscation to reduce the risk of unauthorized access to source code through reverse engineering.

Application Defense

Advances in application architectures and distributed computing have made application components increasingly vulnerable to attack and more difficult to defend on instance-by-instance and session-by-session bases. Dotfuscator CE 5 includes the ability to inject logic that detects the execution of tampered code and, optionally, transmit an incident alert to a developer-specified endpoint and/or terminate the tampered session – all in real-time, across devices, platforms, and distribution models.

Application Measurement

Access to detailed information regarding the behavior of .NET applications once they've been deployed to users – whether in beta, early release or general availability – has been a dream of application developers. Dotfuscator CE 5 can help answer such questions as:

- What applications are my users running, by version?
- What features are being executed and in what order?
- How long do those features take to execute and what is the trend over time?
- Just how stable is the users' experience in the real world?

Traditionally, the impact of gathering this information on end-user performance has meant that few development teams have taken the steps to get access to such information. And when they do, the complexity of maintaining the instrumentation makes it brittle and hard to adapt to changes.

Dotfuscator CE 5 makes it easy for developers, testers and architects to obtain this information with little or no impact on end-user performance. Dotfuscator CE 5 can inject session and feature monitoring down to the method level and transmit runtime data to one or more endpoints. Whether you are managing a beta cycle, a portfolio of web services, or a product evaluation lifecycle, Dotfuscator CE 5 can provide powerful insight into adoption, user behavior, and, ultimately, the business impact of software.

Application Management

Application portfolio management is becoming a critical issue for ISVs and enterprises alike, which means it's becoming an issue for developers. As software becomes easier to build and more difficult to track, organizations are looking for ways to safely and reliably encode business rules and IT policies directly into their software.

The challenge becomes how to deliver on these requirements without the need to make significant changes to the software. To meet this challenge, Dotfuscator CE 5 can enforce end-of-life deadlines and transmit audit streams to track attempted usage on or after an application's expiration (or expiry) date.

Upgrades

Professional developers requiring ongoing support, maintenance and product updates will want to upgrade their copy of Dotfuscator CE 5 to a commercial edition of Dotfuscator that meets the unique needs of their application development organization. For more information on the advanced application security, monitoring, measurement, and management capabilities of the full Dotfuscator product, including a detailed comparison of all editions of Dotfuscator, please visit The Dotfuscator Family Overview. Fully-supported trials of Dotfuscator are available on request.

Commercial versions of Dotfuscator include enhanced security, tighter integration with Visual Studio, and other major extensions to the monitoring and management capabilities of Dotfuscator CE 5. These include:

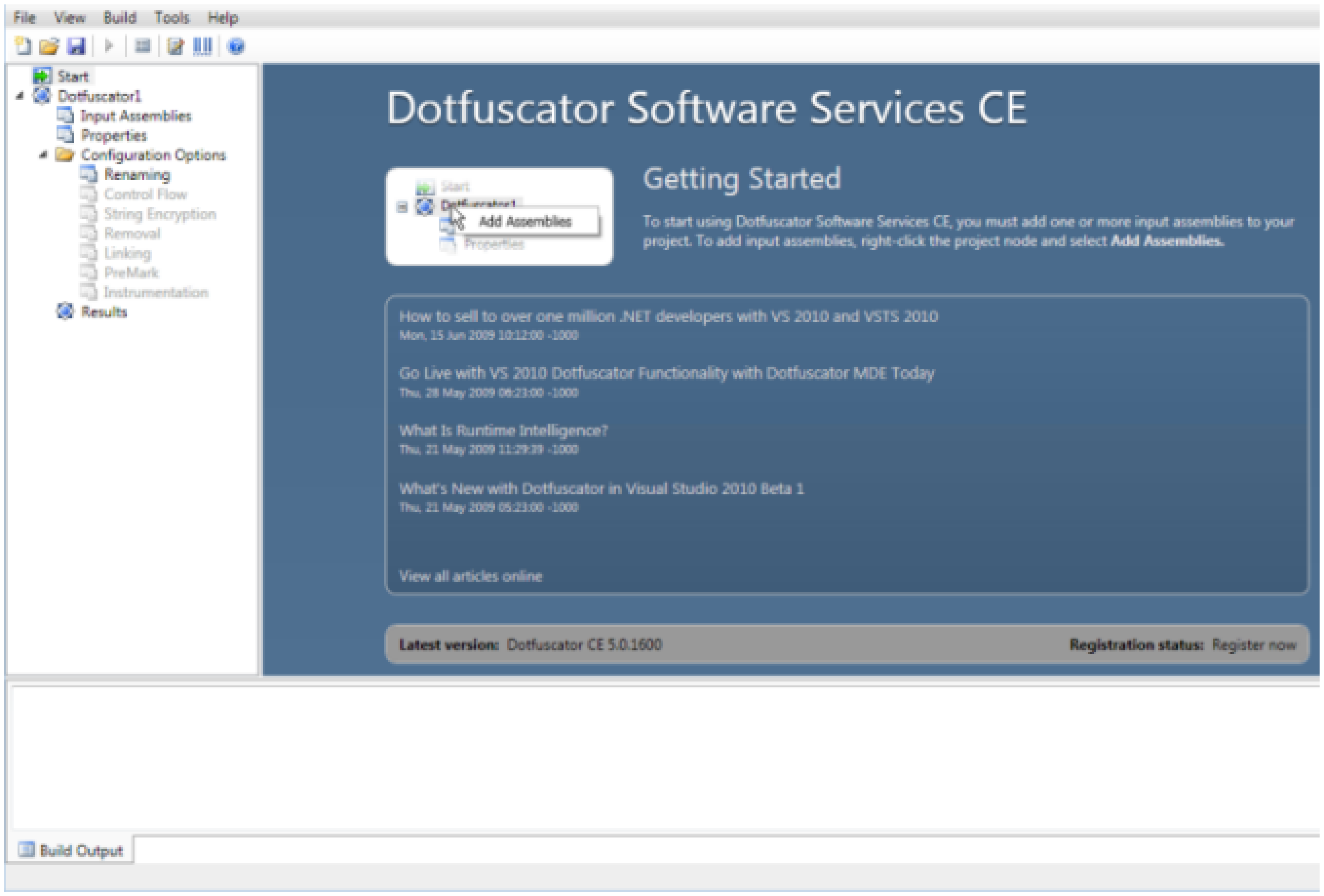
- *Application protection:* Access to the full range of expanded obfuscation transforms, as well as the ability to embed watermarking and the ability to link and/or prune assemblies.
- *Application defense:* The ability to inject custom application defense behaviors and activate tamper incident reporting services.
- *Application measurement:* The ability to create extensible data “signals”, create encrypted transmissions, track an unlimited number of features and methods, and access a rich set of application analytics services.
- *Application management:* The ability to inject custom behaviors, add a warning date, obtain reporting analytics and encrypt transmissions.

Getting Started

To begin using Dotfuscator CE, follow these steps:

- Launch Visual Studio.
- From the Visual Studio menubar, click **Tools > Dotfuscator Software Services Community Edition.**

When Dotfuscator Software Services CE opens, the Start screen displays on your desktop:



Dotfuscator Software Services CE

Getting Started

To start using Dotfuscator Software Services CE, you must add one or more input assemblies to your project. To add input assemblies, right-click the project node and select **Add Assemblies.**

How to sell to over one million .NET developers with VS 2010 and VSTS 2010
Mon, 15 Jun 2009 10:12:00 -1000

Go Live with VS 2010 Dotfuscator Functionality with Dotfuscator MDE Today
Thu, 28 May 2009 06:23:00 -1000

What Is Runtime Intelligence?
Thu, 21 May 2009 11:29:39 -1000

What's New with Dotfuscator in Visual Studio 2010 Beta 1
Thu, 21 May 2009 05:23:00 -1000

View all articles online

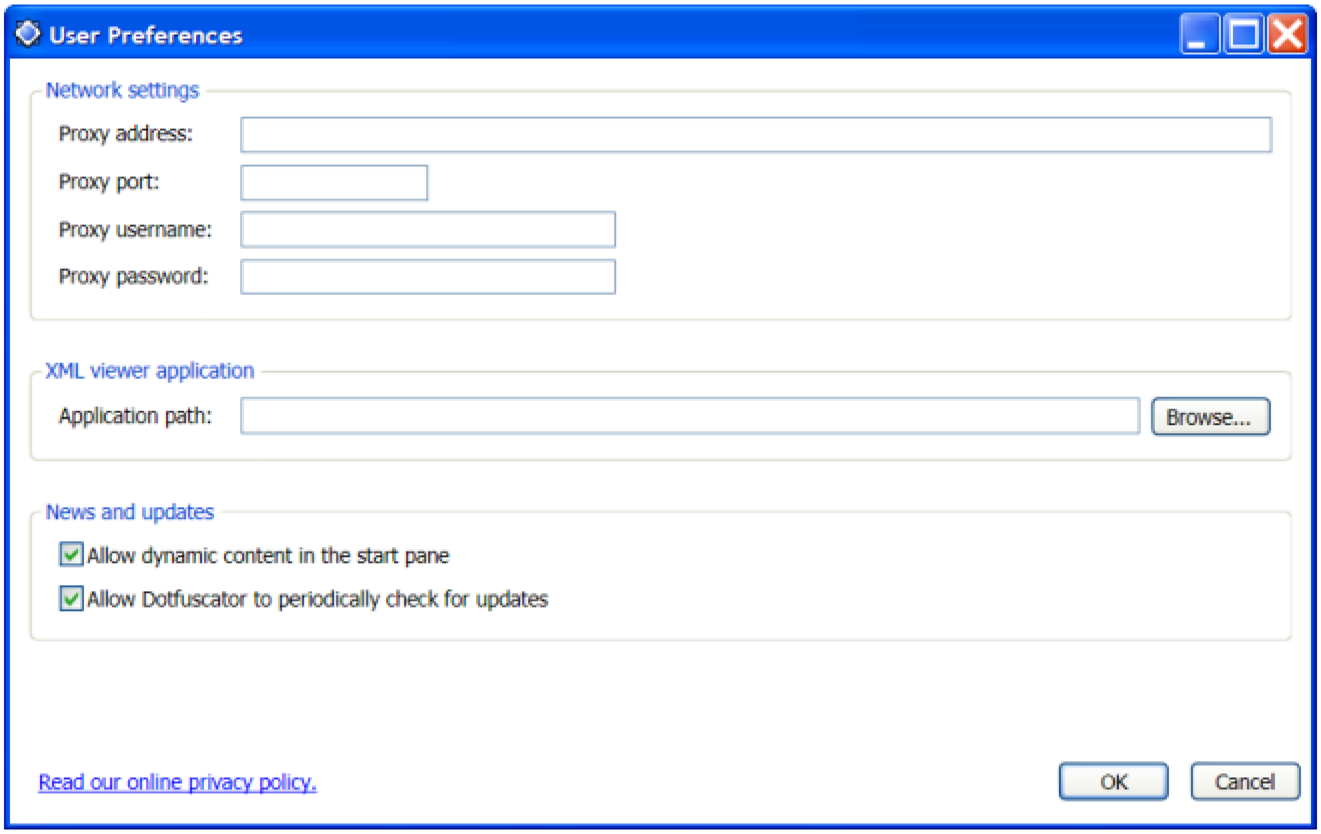
Latest version: Dotfuscator CE 5.0.1600 **Registration status:** Register now

The program consists of three panels: the navigation tree, work area, and build output.

From the Start screen, you may perform several actions. If **Dynamic Content** is enabled, then you can view the release notes about the latest version of Dotfuscator Pro and see the latest news releases from PreEmptive Solutions. If **Dynamic Content** is disabled, then the option to set user preferences displays on the Start screen. The option to register Dotfuscator is visible at all times. Dynamic Content can be enabled or disabled in **User Preferences**.

Setting User Preferences

In the Start screen, if Dynamic Content is disabled, there is a link you may click to set User Preferences, or you may click **Tools > User Preferences**. Regardless of which path you choose to get to User Preferences, this dialog box displays:


User Preferences

Network settings

Proxy address:

Proxy port:

Proxy username:

Proxy password:

XML viewer application

Application path:

News and updates

[✓] Allow dynamic content in the start pane

[✓] Allow Dotfuscator to periodically check for updates

If necessary, enter the configuration details of your network's proxy server requirements in the *Network settings* section of the User Preferences dialog. Proxy information is not required if you do not have a proxy server or if those settings are controlled via Internet Explorer.

If you wish to use an XML viewer for viewing Dotfuscator XML files that is *not* your system's default viewer, you may click **Browse...** in the *XML viewer application* section and locate the application you wish to use.

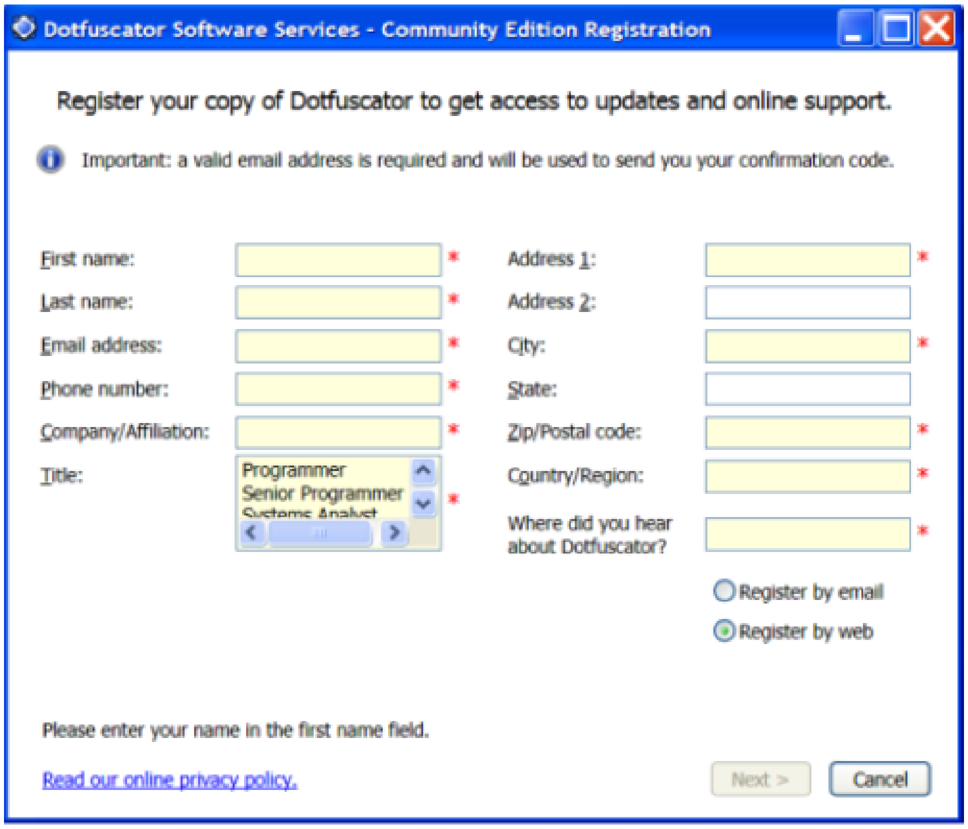
In the *News and updates* section, you may opt to allow Dotfuscator to periodically check for updates. Also, this section is where you can enable Dynamic Content to be displayed in the Start screen.

Latest Version of Dotfuscator

The work area contains a link to the latest release notes for Dotfuscator. If Dynamic Content is disabled, this link will take you to the Dotfuscator version history page at www.preemptive.com.

Registration Status

You can register Dotfuscator CE via the **Register Now** link. Clicking this link displays the registration dialog.


Dotfuscator Software Services - Community Edition Registration

Register your copy of Dotfuscator to get access to updates and online support.

Important: a valid email address is required and will be used to send you your confirmation code.

First name:	*	Address 1:	*
Last name:	*	Address 2:	
Email address:	*	City:	*
Phone number:	*	State:	
Company/Affiliation:	*	Zip/Postal code:	*
Title: Programmer / Senior Programmer / Systems Analyst	*	Country/Region:	*
		Where did you hear about Dotfuscator?	*

Register by email [✓ Register by web]

Please enter your name in the first name field.

Register your product to receive updates and enhancements to Dotfuscator CE. Enter your information within each field. For automated submission, select **Register by web**.

Clicking **Next** takes you to the second page of the Registration dialog. If necessary, enter the configuration details of your network's proxy server requirements. Click **Submit** upon completion.

Once your registration request is received and processed, you will receive a confirmation email from PreEmptive Solutions containing your registration verification code and credentials to the registered support area of www.preemptive.com where you will be able to access updated versions of Dotfuscator CE.

You are now ready to begin obfuscating and instrumenting your application with Dotfuscator CE.

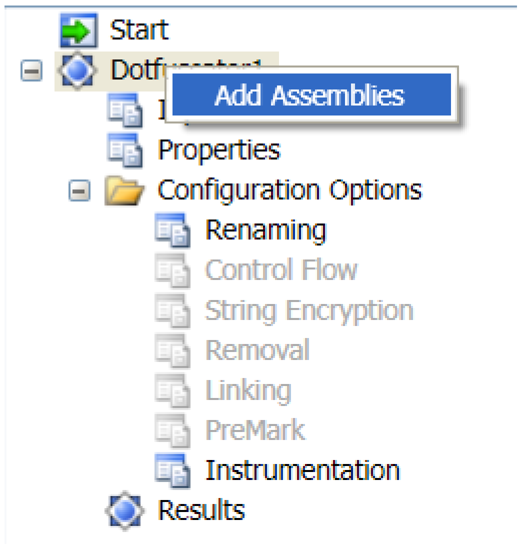
In this section

Input Assemblies

Properties

Input Assemblies

To start using Dotfuscator CE, add one or more input assemblies to your project. To add assemblies to your project, you need to have the Input Assemblies screen open in your work area. You can activate this screen by right-clicking the project node in the navigation tree and selecting **Add Assemblies** or by selecting the Input Assemblies option in the navigation tree.

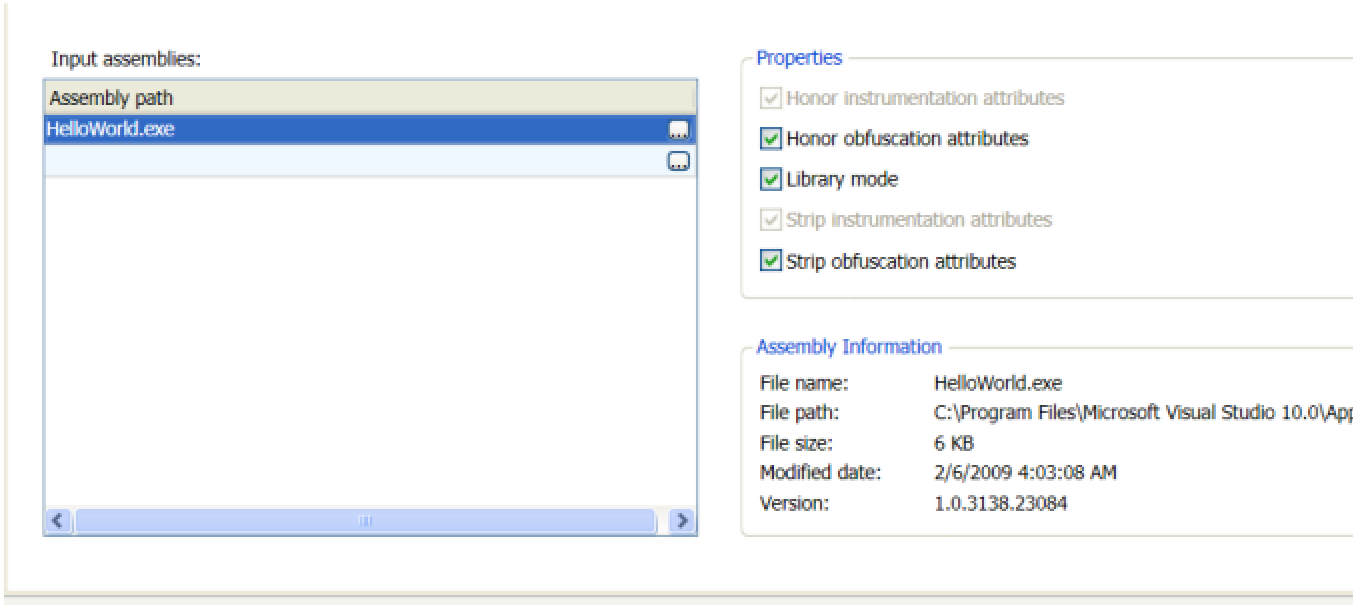


To add your assemblies to the list of Input Assemblies you may drag and drop your .exe and/or .dll files into the *Input assemblies:* section of the page.

You may also click the ... (ellipses) in the *Input assemblies:* section and browse to where your assembly is located and select it.

Additionally, you may enter the path and file name of the assembly in the *Assembly path* field of the *Input assemblies:* section.

Assembly Properties



[✓] Honor instrumentation attributes
[✓] Honor obfuscation attributes
[✓] Library mode
[✓] Strip instrumentation attributes
[✓] Strip obfuscation attributes

File name:	HelloWorld.exe
File path:	C:\Program Files\Microsoft Visual Studio 10.0\Ap
File size:	6 KB
Modified date:	2/6/2009 4:03:08 AM
Version:	1.0.3138.23084

In the *Properties* section of the work area panel, select or de-select:

Honor instrumentation attributes: Selecting *Honor instrumentation attributes* tells Dotfuscator to process these attributes and perform the indicated instrumentation transformations on the target assembly. De-selecting this option tells Dotfuscator to ignore any instrumentation attributes.

Instrumentation attributes are custom attributes that can be applied in your source code to track application stability, features, usage, and to add shelf life functionality.

Honor obfuscation attributes: Selecting *Honor obfuscation attributes* tells Dotfuscator to process these attributes and perform the indicated obfuscation transformations on the target assembly. De-selecting this option tells Dotfuscator to ignore any obfuscation attributes.

Obfuscation attributes are custom attributes that can be applied in your source code to explicitly declare the inclusion or exclusion of types, methods, enums, interfaces, or members from various types of obfuscation. The attribute you would use to include or exclude types, methods, enums, interfaces, and members from obfuscation is **System.Reflection.ObfuscationAttribute**. If you want to denote that a specific assembly will have its items included or excluded from obfuscation, you would use **System.Reflection.ObfuscateAssemblyAttribute**. Dotfuscator CE only supports renaming. Additional obfuscation functionality is available in the commercial version of Dotfuscator.

Library mode: This setting tells Dotfuscator that the selected input assembly constitutes a library. For obfuscation purposes, a library is defined as an assembly that is referenced from other components not specified as one of the inputs in this run. When an assembly is obfuscated in Library Mode, publicly visible items are not renamed, thus keeping your public API accessible to external callers.

Strip obfuscation attributes: Dotfuscator can strip out all of the obfuscation attributes when processing is complete, so output assemblies will not contain clues about how it was obfuscated. Selecting this option tells Dotfuscator to remove these attributes from the target output assembly. De-selecting this option tells Dotfuscator to leave the attributes in the output assembly unless the individual attributes designate that they should be stripped via the StripAfterObfuscation property.

Assembly Information

This section of the Input Assemblies work area provides general data about the currently selected assembly, including the file name, file path, file size, (last) modified date, and the version.

Properties

The **Properties** item on the navigation tree brings up the Properties configuration in the work area. It contains six tabs, three of which are available to Dotfuscator CE users.

Project Properties

Project Properties can be thought of as simple string substitution macros that may be used wherever a filename or path is required. The **Project Properties** screen is where you can view, delete, and manually add user-defined name/value pairs as Project Properties and to view External Properties. External properties are those either defined by Dotfuscator (e.g. configdir, appdatadir) or those passed on the command line via the /p switch. Project Properties are especially useful in creating obfuscation build templates to support multiple projects and configurations. Properties are referenced with the following syntax:

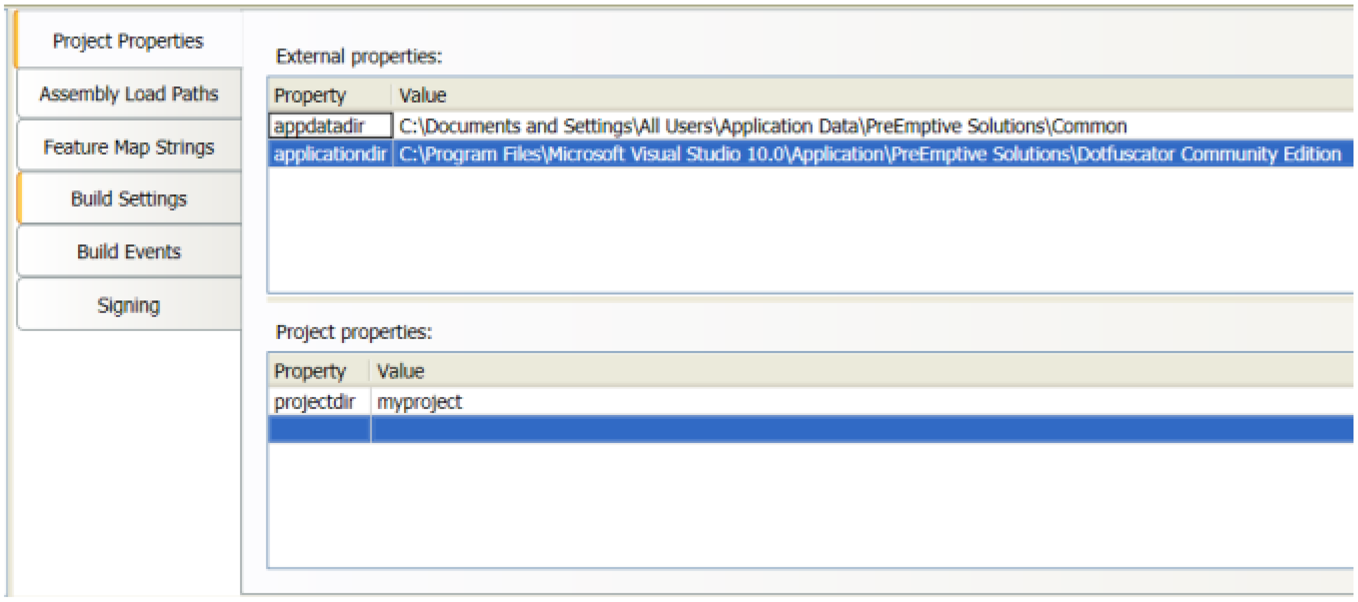
Property Reference Syntax

`${property_name}`

Property references are case sensitive, therefore `${MyProjectDir}` references a different property than does `${myprojectdir}`. Property references are interpreted literally and may not be nested. Currently, property references may only be used as values in the `dir` or `name` attributes of the `<file>` element.

Dotfuscator uses the following algorithm to find a value associated with the property:

- Check the external property list for a value.
- If not found, check for an environment variable with the same name as the property.
- If not found, check for a project property.
- If still not found, use the empty string as the value.



External properties:

Property	Value
appdatadir	C:\Documents and Settings\All Users\Application Data\PreEmptive Solutions\Common
applicationdir	C:\Program Files\Microsoft Visual Studio 10.0\Application\PreEmptive Solutions\Dotfuscator Community Edition

Project properties:

Property	Value
projectdir	myproject

External Properties

The *Property* column contains the name of the property, and the *Value* column contains the value of that property. Dotfuscator CE defines the following three Properties and Values:

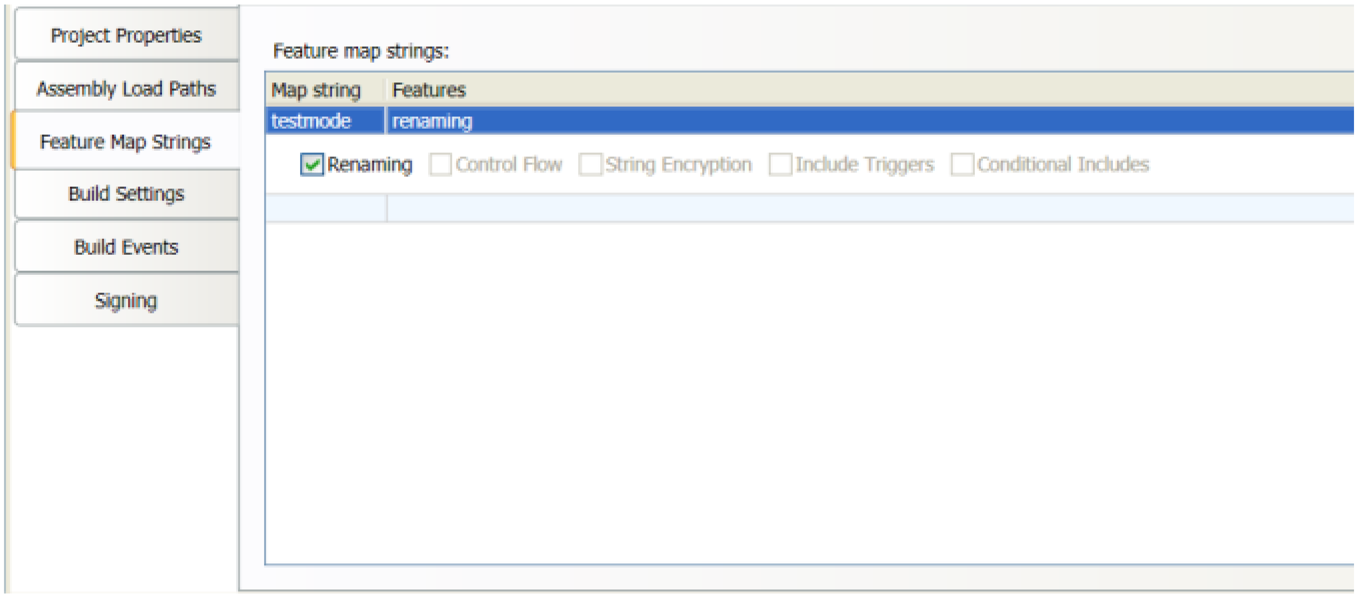
- `applicationdir` reflects Dotfuscator's installation directory where the Dotfuscator application files are located.
- `appdatadir` reflects Dotfuscator's local data directory.
- `configdir` reflects the directory in which the current project's configuration file resides.

Project Properties

This section allows for the definition and assignment of arbitrary additional properties that will be used during the build process. These properties will be evaluated in the same manner as the external properties. In the above example, `projectdir` is the *Property* and `myproject` is the *Value* of the Property.

Feature Map Strings

The Feature Map Strings section is used for Declarative Obfuscation. Declarative Obfuscation is implemented via attribute decoration within the source code while it is being written. The attributes that are used to control Declarative Obfuscation are **System.Reflection.ObfuscateAssemblyAttribute** and **System.Reflection.ObufscationAttribute**. System.Reflection.ObfuscateAssemblyAttribute controls the obfuscation of the assembly as a whole. System.Reflection.ObufscationAttribute controls the obfuscation of individual types and their members. Feature Map Strings enable you to declare, within the source code, what should and should not be obfuscated by using attributes.



Map string	Features
testmode	renaming

[✓ Renaming] Control Flow String Encryption Include Triggers Conditional Includes

In this section, you can add, edit, and remove feature map strings. To add a map string, click in the *Map String* field and type the **name** of the string. When you do this, a row of checkboxes displays beneath the current row containing the feature you can select. For example,

- `testmode` is the **name** of the *Map string* that was entered
- `renaming` is the **Feature** of the Map string that was selected from the row beneath.

In Dotfuscator CE, renaming is the only available feature. Commercial versions of Dotfuscator support additional features as follows:

Feature String	Action
***controlflow**	attribute configures control flow obfuscation.
***stringencryption**	attribute configures string encryption
***trigger**	attribute configures pruning by marking the annotated item as an entry point
***conditionalinclude**	attribute configures pruning by conditionally including the annotated item

By decorating items with an obfuscation attribute and ensuring that their input assemblies are set to honor obfuscation attributes, you can specify which items in you application should not be renamed. In the following code sample your method is invoked via reflection and should be excluded from renaming.

Invoked via Reflection, Excluded from Renaming:

```
[System.Reflection.Obfuscation(Exclude=true, Feature="renaming")]
public void CalledFromReflection(int someValue) {
```

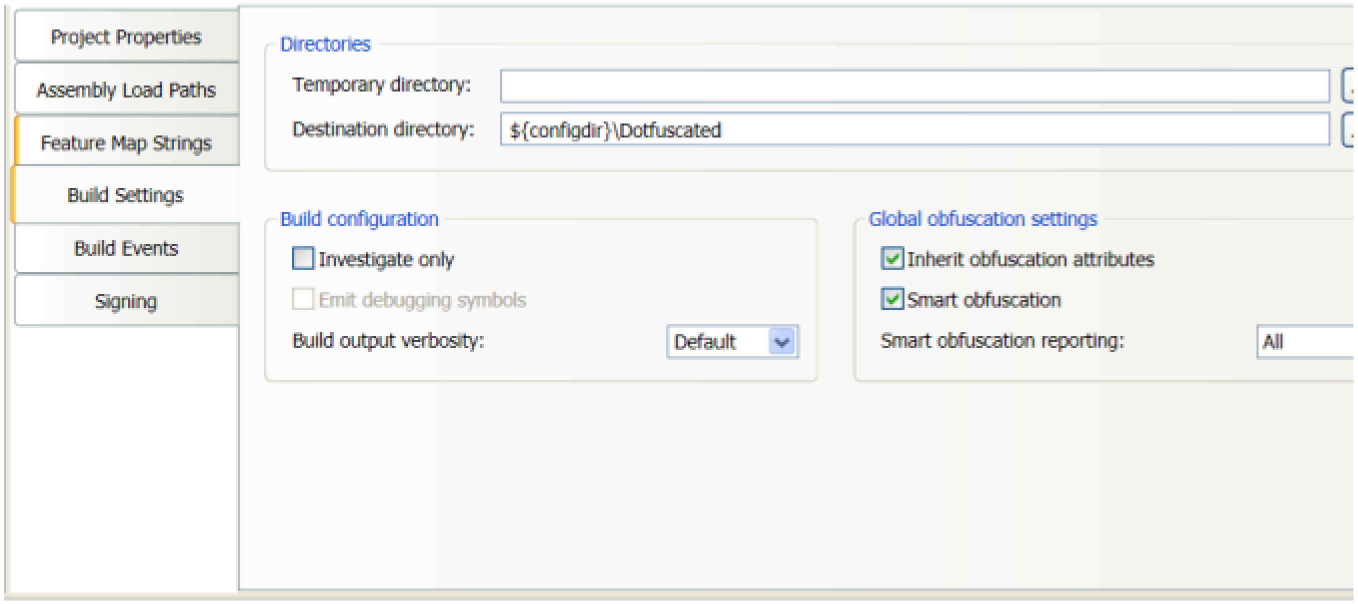
If you need to exclude a method from obfuscation in a specific build configuration, then create a Feature Map declaration and set the feature value of the obfuscation attribute to match. In the screenshot above, we have set a Feature Map string of "testmode" that will be used to exclude items from being renamed. In the code sample below, we have decorated our method so that it will be renamed by Dotfuscator unless the "testmode" feature string is defined.

Testmode Feature Map Declaration:

```
[System.Reflection.Obfuscation(Exclude=true, Feature="testmode")]
public void OnlyCalledFromReflectionInTestMode(int someValue){
```

Build Settings

This is where you set the destination directory, build configuration, and global obfuscation settings.



Temporary directory:
Destination directory: ${configdir}\Dotfuscated
Investigate only [ ]
Emit debugging symbols [ ]
Build output verbosity: Default
Inherit obfuscation attributes [✓]
Smart obfuscation [✓]
Smart obfuscation reporting: All

Directories

The *Temporary directory:* is optional and is used to store temporary files during processing. By default Dotfuscator CE uses your Windows Temporary directory. If you wish to specify this directory, enter the **name** of the directory in this field. For example, you would enter `${projectdir}\temp` in this field, or click **Browse** to determine its location.

The *Destination directory:* is required as that is where the output from the build will reside. For example, you would type `${projectdir}\output` in this field, or click **Browse** to determine its location.

Build Configuration

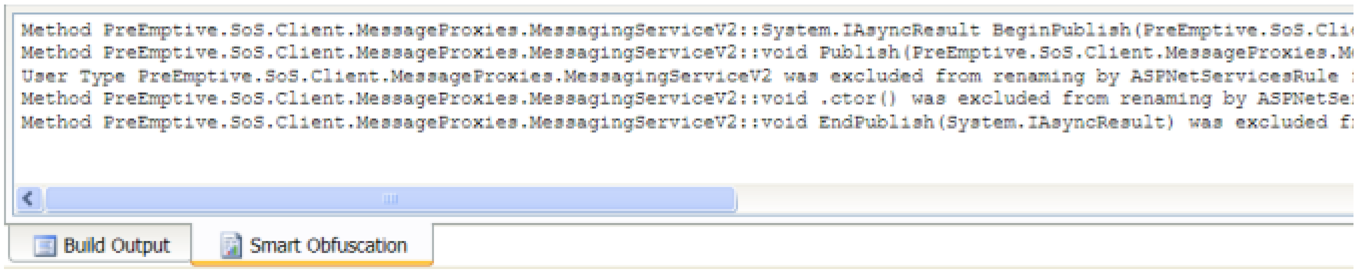
Selecting *Investigate only* shows you what will occur as a result of processing the input assemblies, without actually writing output assemblies. This is useful for creating map files and reports.

Build output verbosity: enables you to determine the amount of information presented to you in the **Build Output** panel. Selecting **Quiet** keeps the output information minimal or non-existent. Selecting **Verbose** provides you detailed information about all actions occurring during the build process. The Verbose option is especially helpful if you are attempting to fine-tune your results.

Global Obfuscation Settings

In this section, you can select or deselect *Inherit obfuscation attributes* and *Smart obfuscation*. *Inherit obfuscation attributes* specifies whether an obfuscation attribute that is put on a type will also be applied to derived types. *Smart Obfuscation* uses static analysis to determine what elements should be excluded from renaming. Sometimes a rule can recognize that an action needs to be taken, but cannot determine what specific action to take because static analysis does not yield enough information. When this happens, the rule issues a warning.

You can also determine if you want Smart obfuscation reporting by selecting **All** for all messages, **None** for no messages, or **Warnings only**. When Smart Obfuscation is enabled, and the project is built, a Smart Obfuscation report will be presented in its own tab near the Build Output tab and if any Smart Obfuscation warnings are generated, they will be presented in the Smart Obfuscation Warnings tab.



```
Method PreEmptive.SoS.Client.MessageProxies.MessagingServiceV2::System.IAsyncResult BeginPublish(PreEmptive.SoS.Cli
Method PreEmptive.SoS.Client.MessageProxies.MessagingServiceV2::void Publish(PreEmptive.SoS.Client.MessageProxies.M
User Type PreEmptive.SoS.Client.MessageProxies.MessagingServiceV2 was excluded from renaming by ASPNetServicesRule
Method PreEmptive.SoS.Client.MessageProxies.MessagingServiceV2::void .ctor() was excluded from renaming by ASPNetSe
Method PreEmptive.SoS.Client.MessageProxies.MessagingServiceV2::void EndPublish(System.IAsyncResult) was excluded f
```

Assembly Load Path, Build Events, and Signing

These features are exclusive to Dotfuscator Professional Edition. To request information and a free 14-day evaluation (where applicable), click the Try Now button on the toolbar. To learn how to purchase Dotfuscator Professional Edition, click the Buy Now button on the toolbar. Alternately, you may select these options from the Help menu.

Configuration Options

The Configuration Options section is where you can set renaming options and exclusions, choose built-in renaming rules, and configure instrumentation settings.

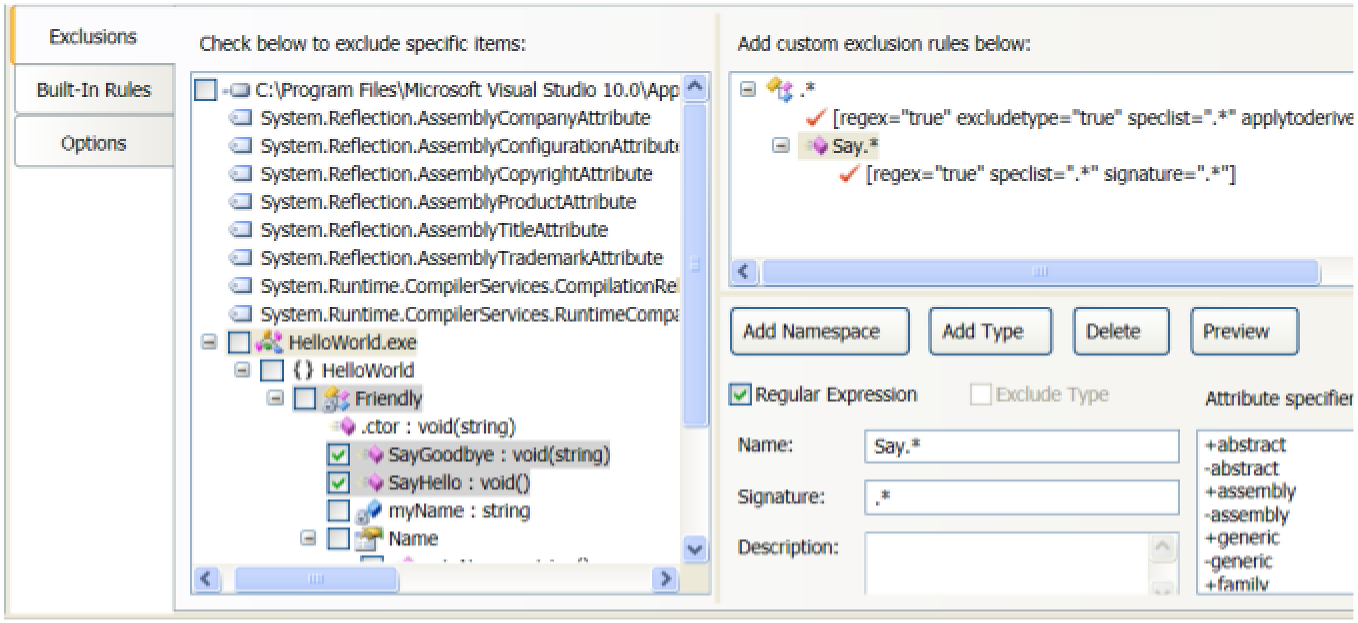
Options such as Control Flow Obfuscation, String Encryption, Removal, Linking, and PreMark are exclusive to Dotfuscator Professional Edition. To request information and a free 14-day evaluation (where applicable) click the Try Now button on the toolbar. To learn how to purchase Dotfuscator Professional Edition, click the Buy Now button on the toolbar. Alternately, you may select these options from the Help menu.

In this section

Renaming

The Renaming editor displays three configuration tabs: the **Exclusions** tab, which is used to graphically set custom exclusion rules; the **Options** tab, which is used to configure other options related to renaming; and the **Built-in Rules** tab, which displays pre-configured renaming exclusion rules that apply to specific application types or technologies.

Renaming Exclusions



```
☐ C:\Program Files\Microsoft Visual Studio 10.0\App
    System.Reflection.AssemblyCompanyAttribute
    System.Reflection.AssemblyConfigurationAttribute
    System.Reflection.AssemblyCopyrightAttribute
    System.Reflection.AssemblyProductAttribute
    System.Reflection.AssemblyTitleAttribute
    System.Reflection.AssemblyTrademarkAttribute
    System.Runtime.CompilerServices.CompilationRe
    System.Runtime.CompilerServices.RuntimeCompa
  ⊟ ☐ HelloWorld.exe
    ⊟ ☐ {} HelloWorld
      ⊟ ☐ Friendly
            .ctor : void(string)
          ☑ SayGoodbye : void(string)
          ☑ SayHello : void()
          ☐ myName : string
        ⊟ ☐ Name
```

```
⊟ .*
    ✓ [regex="true" excludetype="true" speclist=".*" applytoderive
  ⊟ Say.*
      ✓ [regex="true" speclist=".*" signature=".*"]
```

☑ Regular Expression ☐ Exclude Type

Name:	Say.*
Signature:	.*
Description:	

+abstract
-abstract
+assembly
-assembly
+generic
-generic
+family

The **Renaming Exclusions** tab lets you determine which parts of your program are to be excluded from the renaming process. You may exclude specific items from renaming by browsing the navigation tree and checking those items.

The Rename *Exclusions* Tab gives you complete granular control over all parts of your program that you may wish to exclude from the renaming process.

You may exclude specific items from renaming by browsing the tree view of your application and checking the items you want to exclude. In addition, you may visually create your own custom rules for selecting multiple items for exclusion.

Custom rules can be defined to exclude larger selections of items without requiring you to individually select them for exclusion in the treeview. Rules can be defined to exclude namespaces, classes, methods, and fields based on their name. Additionally, a Regular Expression can be used to provide a way to specify a pattern with any items that match that pattern from being excluded from obfuscation.

To help you fine-tune your exclusion rules, you can preview their cumulative effects at any time by clicking the **Preview** button. The application tree view shades all items selected for exclusion.

To preview a specific rule, right click on the rule whose effects you wish to see, select **Preview**, and Dotfuscator highlights items that will be excluded due to that rule.

Excluding a namespace will exclude all types and their members in the specified namespace. To exclude a namespace select the **Add Namespace** button and enter the name of the namespace you wish to exclude in the *Name* field. If you wish to use a regular expression to match multiple namespaces ensure that the *Regular Expression* box is checked and that the *Name* field contains the regular expression you wish to match against.

Types can be excluded by either matching only on their names or by optionally including a set of attribute specifiers that will further restrict the types excluded from renaming. Nested classes can be excluded by using a backslash "\" as a delimiter between the outer and inner class. By checking the *Regular Expression* box the value in the *Name* field will be matched as a regular expression rather than a literal string when determining which items to exclude from obfuscation. Type exclusion can follow inheritance rules. If the *Apply to Derived Types* option is selected then the exclusion rule will additionally be applied to the matched types and any types that inherit from them.

A number of other exclusion options are available and are defined as child rules to a containing type rule. To create a child rule right click on the type rule you wish to enhance and select the particular child rule implementation you wish to add.

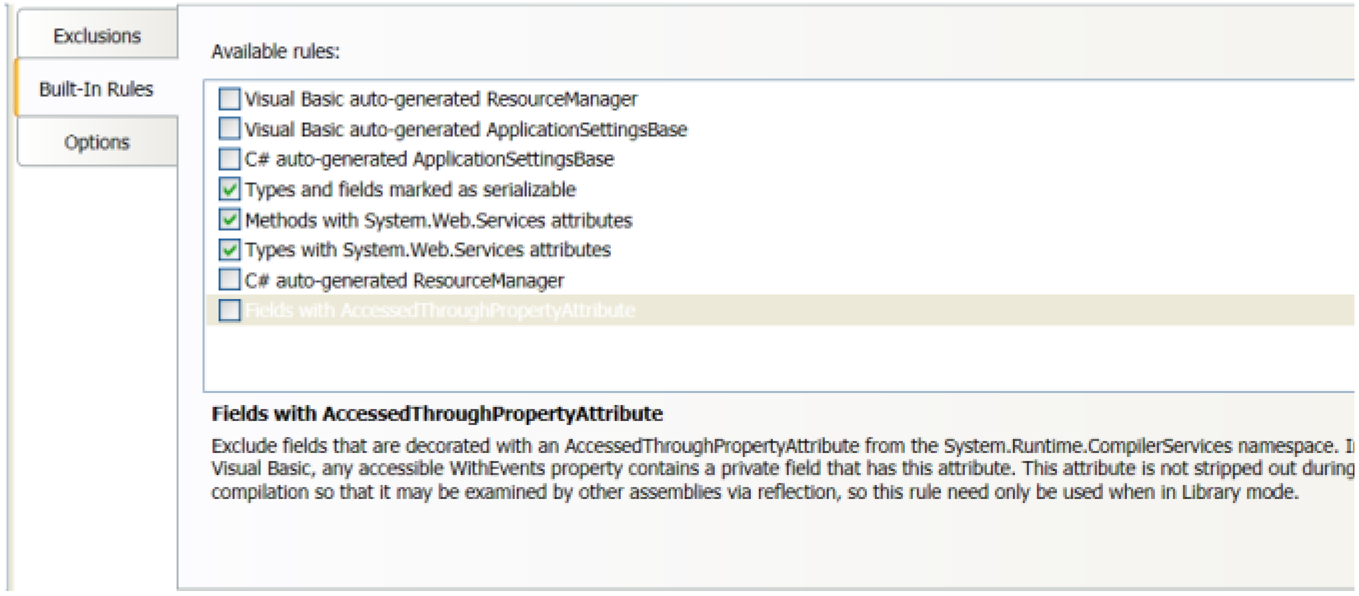
Rules to exclude methods, fields, properties, and events are set by creating a type rule for the type(s) that contain those items and adding a child rule for the appropriate item. As with namespaces and types, the exclusion of other items can be based on their literal name or a regular expression.

Types, methods, fields, and properties can be selected for exclusion by annotating them with custom attributes, then creating a type exclusion rule containing a **CustomAttribute** child rule. Matching on the name of the CustomAttribute is used to determine if an item should be excluded and regular expression matching logic will be implemented if the *Regular Expression* option is selected. Exclusion by custom attribute can also be propagated down the inheritance hierarchy by using the *Allow Inheritance* option which additionally excludes any subtypes or overriding methods and properties with the specified attribute.

Supertypes can also be excluded from obfuscation by adding a supertype child rule to a type rule definition. An item is excluded from obfuscation if it inherits from one or more supertypes that match the supertype exclusion condition.

In order to specify that an exclusion rule apply only to a method, field, property, event, supertype, or custom attribute and to include the containing type in obfuscation, select the *Exclude Type* option when defining the type rule. This option leaves the type included for renaming purposes and only evaluates the child rules.

Renaming Built-in Rules



Available rules:

- [ ] Visual Basic auto-generated ResourceManager
- [ ] Visual Basic auto-generated ApplicationSettingsBase
- [ ] C# auto-generated ApplicationSettingsBase
- [✓] Types and fields marked as serializable
- [✓] Methods with System.Web.Services attributes
- [✓] Types with System.Web.Services attributes
- [ ] C# auto-generated ResourceManager
- [ ] Fields with AccessedThroughPropertyAttribute

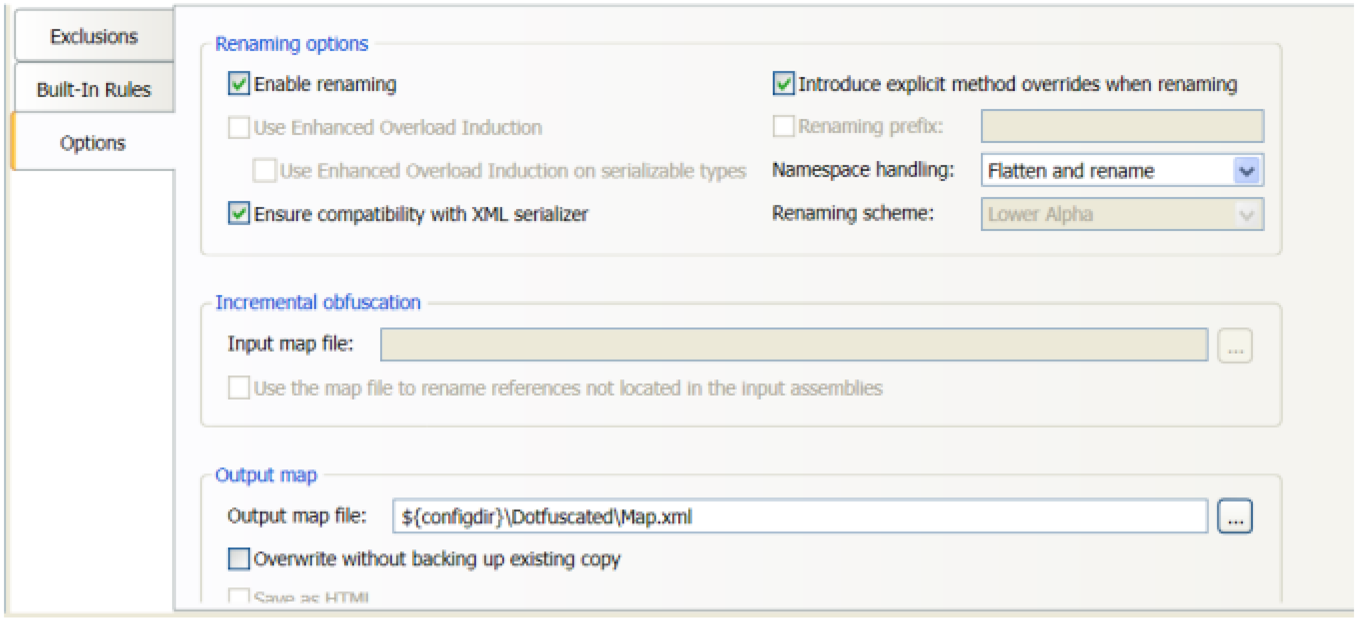
Fields with AccessedThroughPropertyAttribute

Exclude fields that are decorated with an AccessedThroughPropertyAttribute from the System.Runtime.CompilerServices namespace. I
Visual Basic, any accessible WithEvents property contains a private field that has this attribute. This attribute is not stripped out during
compilation so that it may be examined by other assemblies via reflection, so this rule need only be used when in Library mode.

The **Renaming Built-in Rules** tab displays pre-configured renaming exclusion rules that apply to specific application types or technologies. Each rule has a description that displays on the screen when the rule is selected. You can apply a built-in rule to your project by checking it.

Renaming Options

The **Renaming Options** tab is used to set renaming options and to identify map file output options. The *Renaming Options* section contains configuration settings that control how renaming is performed.



Renaming options

- [✓] Enable renaming
- [ ] Use Enhanced Overload Induction
- [ ] Use Enhanced Overload Induction on serializable types
- [✓] Ensure compatibility with XML serializer
- [✓] Introduce explicit method overrides when renaming
- [ ] Renaming prefix:
- Namespace handling: Flatten and rename
- Renaming scheme: Lower Alpha

Incremental obfuscation

- Input map file:
- [ ] Use the map file to rename references not located in the input assemblies

Output map

- Output map file: ${configdir}\Dotfuscated\Map.xml
- [ ] Overwrite without backing up existing copy
- [ ] Save as HTML

- **Enable renaming** - Select this to enable renaming. You may also right click on **Renaming** in the navigation tree and check **Enable** to enable renaming. Likewise, if renaming is enabled, you may disable it by right clicking on Renaming in the navigation tree and unchecking Enable. If this option is unchecked, Dotfuscator will not perform renaming when your project is built.
- **Ensure compatibility with XML serializer -** Select this to configure renaming in such a way as to allow XML serialization.
- **Introduce explicit method overrides when renaming** - Select this to allow overriding methods to have different names from those of the methods they override.

You may also select the appropriate *Namespace handling* options:

- **Flatten and rename** removes the namespaces.
- **Rename only** renames the namespaces, but keeps the hierarchy.
- **Preserve** keeps the namespaces exactly as they are in the input assembly.

The *Output map* section is where the destination of the output map file is set. You also have the option of overwriting the output file each time you build the application without generating a backup of any existing map files.

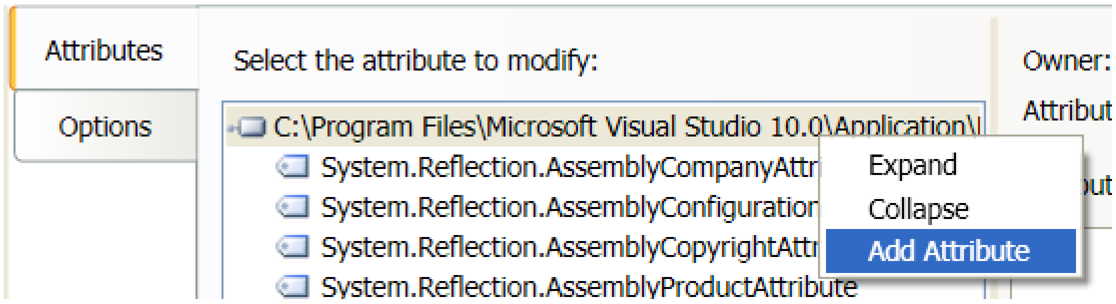
Instrumentation

Instrumentation adds pre-built application usage tracking, binary tamper detection, and application expiration to your applications without requiring any additional code. Dotfuscator injects the required code for you. You can define injection points either via custom attributes in your source code or via extended attributes specified within the Dotfuscator user interface.

The Instrumentation editor allows you to add, edit, and review custom and extended attributes. In Dotfuscator CE, the Instrumentation editor displays two configuration tabs: the **Attributes** tab and the **Options** tab. The Attributes tab is where you add new extended attributes to methods or modify custom attributes that are already present. The **Options** tab is where you select which types of messages - application analytics, shelf life notification, and tamper alert - you want to send to the Runtime Intelligence Service.

Assembly Attributes

Assembly attributes are attributes that may only be applied at the assembly level. Assembly attributes are used by the code injection process to add unique identification data to an assembly, such as company information.


Attributes
Options
Select the attribute to modify:
C:\Program Files\Microsoft Visual Studio 10.0\Application\
System.Reflection.AssemblyCompanyAttr
System.Reflection.AssemblyConfiguration
System.Reflection.AssemblyCopyrightAttr
System.Reflection.AssemblyProductAttribute
Expand
Collapse
Add Attribute
Owner:
Attribut

Required Assembly Attributes

To instrument your application for Runtime Intelligence Services, there are two required assembly attributes. These attributes are added to the application by right clicking on an assembly node in the *Select the attribute to modify:* tree.

BusinessAttribute

The BusinessAttribute is required and contains values that are used to identify the company that owns the application being instrumented. This attribute consists of a *CompanyKey*, which is a unique value used to identify the creator of the application and is required for the login to the free Runtime Intelligence Services Portal. For convenience, you can generate a new unique identifier by pressing the "**...**" button associated with the CompanyKey entry area. For optimal usage, you should generate a single unique CompanyKey that you use for all of your applications. The *CompanyName* may be left empty; however, you are encouraged to enter the name of the company for Runtime Intelligence Services Portal personalization.

ApplicationAttribute

The ApplicationAttribute is required and contains values that are used to identify the instrumented application:

- **ApplicationType**. This identifies the type of application being instrumented such as a Windows application.
- **GUID**. This is a unique identifier for the application that is generated by clicking the "..." in this field.
- **Name**. This is the name of the application. If you leave this blank, the name will be filled in via reflection (if allowed at runtime). Only fill it in if you want a name in the portal that is different from the name in the assembly, or if reflection can't be used by your application at runtime (e.g. it is a Silverlight application).
- **Version**.This is the version of the application. If you leave this blank, the name will be filled in via

reflection (if allowed at runtime). Only fill it in if you want a version in the portal that is different from the name in the assembly, or if reflection can't be used by your application at runtime (e.g. it is a Silverlight application).

BinaryAttribute

The BinaryAttribute contains a value that is used to identify the specific assemblies that make up the instrumented application. The *GUID* is the unique identifier for the assembly and is used to specify which assembly was altered when tampering was detected. Although this attribute is not required, if you are using tamper detection, you should set a binary attribute on each assembly so the notifications can include the assembly information.

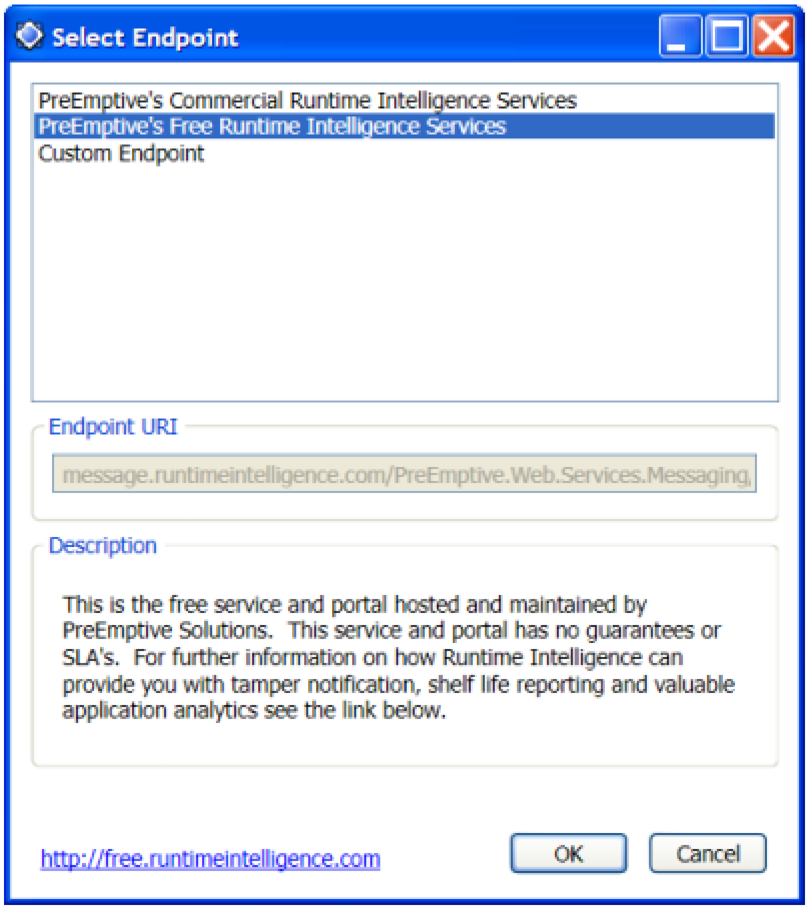
Functional Attributes

Functional Attributes can be used to track application stability, feature usage, tampering attempts, and the frequency of an application's use. They may also be used to make an application cease to function after a specified date.

SetupAttribute

The SetupAttribute is used to track when an application has started. Dotfuscator injects startup code for message sending and sends the startup message. There must be one or more methods with this attribute in an assembly or application that uses Runtime Intelligence. This attribute should be placed as closely as possible to the normal application startup logic. At runtime, the functionality generated by this attribute will begin sending usage data back to the Runtime Intelligence Services Portal. Optionally, you may specify the location in your code of functionality that controls the transmission of messages (opt-in or opt-out behavior). Dotfuscator will use this information when generating the initialization code.

When you add the SetupAttribute, you may set the **Custom Endpoint** destination of the messages.The endpoint options include the free Runtime Intelligence Services Portal as a default, the commercially available Runtime Intelligence Services Portal, or a custom endpoint of your own choosing.


Select Endpoint

PreEmptive's Commercial Runtime Intelligence Services
PreEmptive's Free Runtime Intelligence Services
Custom Endpoint

Endpoint URI

message.runtimeintelligence.com/PreEmptive.Web.Services.Messaging,

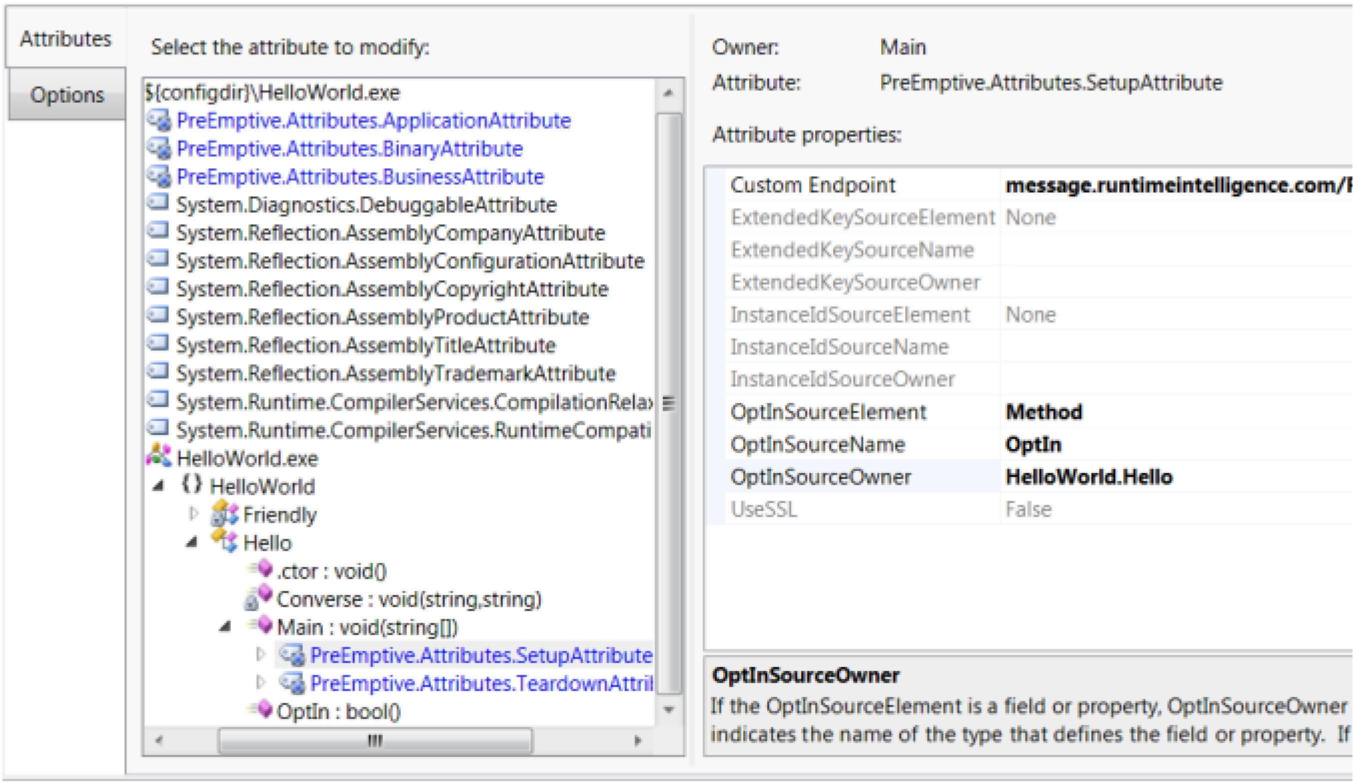
Description

This is the free service and portal hosted and maintained by PreEmptive Solutions. This service and portal has no guarantees or SLA's. For further information on how Runtime Intelligence can provide you with tamper notification, shelf life reporting and valuable application analytics see the link below.

http://free.runtimeintelligence.com

OK Cancel

The SetupAttribute also provides the ability to implement either opt-in or opt-out functionality within your application. This functionality enables your users to elect to provide their usage data or not. To provide the setting determining if execution and usage data will be collected and sent to the Runtime Intelligence Services Portal, you can specify either a method, method argument, field, or property containing or returning a boolean value. If the value is **true** then the application will transmit usage data as configured. If the value is **false** then no usage data will be transmitted by the application.



${configdir}\HelloWorld.exe
PreEmptive.Attributes.ApplicationAttribute
PreEmptive.Attributes.BinaryAttribute
PreEmptive.Attributes.BusinessAttribute
System.Diagnostics.DebuggableAttribute
System.Reflection.AssemblyCompanyAttribute
System.Reflection.AssemblyConfigurationAttribute
System.Reflection.AssemblyCopyrightAttribute
System.Reflection.AssemblyProductAttribute
System.Reflection.AssemblyTitleAttribute
System.Reflection.AssemblyTrademarkAttribute
System.Runtime.CompilerServices.CompilationRela
System.Runtime.CompilerServices.RuntimeCompati
HelloWorld.exe
HelloWorld
Friendly
Hello
.ctor : void()
Converse : void(string,string)
Main : void(string[])
PreEmptive.Attributes.SetupAttribute
PreEmptive.Attributes.TeardownAttri
OptIn : bool()

Owner: Main
Attribute: PreEmptive.Attributes.SetupAttribute

Attribute properties:

Custom Endpoint	**message.runtimeintelligence.com/**
ExtendedKeySourceElement	None
ExtendedKeySourceName	
ExtendedKeySourceOwner	
InstanceIdSourceElement	None
InstanceIdSourceName	
InstanceIdSourceOwner	
OptInSourceElement	**Method**
OptInSourceName	**OptIn**
OptInSourceOwner	**HelloWorld.Hello**
UseSSL	False

OptInSourceOwner
If the OptInSourceElement is a field or property, OptInSourceOwner indicates the name of the type that defines the field or property.

To configure the opt-in/opt-out behavior you must configure the **OptInSourceElement** property with the appropriate type of code element that will contain the Boolean value. The **OptInSourceName** must be the name of the element that contains the Boolean value to be used to determine if Runtime Intelligence data will be collected and transmitted. At runtime, the source value should be set by your application code before the setup method is called. The **OptInSourceOwner** property may be left off if the element is defined in the same class as the method attributed with the SetupAttribute. If the **OptInSourceElement** is a method argument then it must correspond to a method parameter on the method that is annotated with the SetupAttribute.

TeardownAttribute

The TeardownAttribute is used to track when an application has successfully stopped. Dotfuscator injects teardown code for message sending and sends the shutdown message. There must be one or more methods with this attribute in an assembly or application that uses Runtime Intelligence instrumentation. The teardown method does not necessarily have to be the last method called, but it should have the property that it is executed exactly once when the application shuts down, as close as possible to exit.

FeatureAttribute

The FeatureAttribute is for tagging features you wish to track using the Runtime Intelligence Service. This attribute may be placed on any method or property. Runtime Intelligence tracks features using the name provided in this property. Multiple methods can be given the same name which will cause the same usage indicator on the Runtime Intelligence Services Portal to be updated. A feature can be either an atomic event or the execution duration can be measured. To simply track that a feature has been executed, use a FeatureEventType of 'Tick.' To measure the execution duration of the feature, use two feature attributes with the same name, one with a FeatureEventType of 'Start' and the second with a FeatureEventType of 'Stop.' In the Feature Scorecard report on the Portal, you will see the minimum, maximum, and average durations for each feature execution.

InsertShelfLifeAttribute

Add an InsertShelfLifeAttribute to any property or method where you would like an application expiration check to occur. At runtime, your application can send an expiration notification message to the Runtime Intelligence Services Portal if the application is executed after the expiration date has occurred. Upon expiration, the default behavior is to send the expiration notification message; however, you may set the ExpirationNotificationSinkElement property to DefaultAction, which will optionally send an expiration message to the Runtime Intelligence Services Portal and cause the application to immediately exit. In order for the expiration notification messages to be sent, the application must contain methods marked with a Setup and Teardown attribute.

A Shelf Life Activation Key (SLAK) is required to instrument your application with this feature. To obtain a SLAK, please go to our Support site: **http://www.preemptive.com/support/index.html**.

InsertSignofLifeAttribute

The InsertSignofLifeAttribute sends a message each time the method tagged with this attribute is called. At runtime, the sign of life code sends a message indicating that the application has been executed. In order to use this attribute, the application must contain methods marked with a Setup and Teardown attribute.

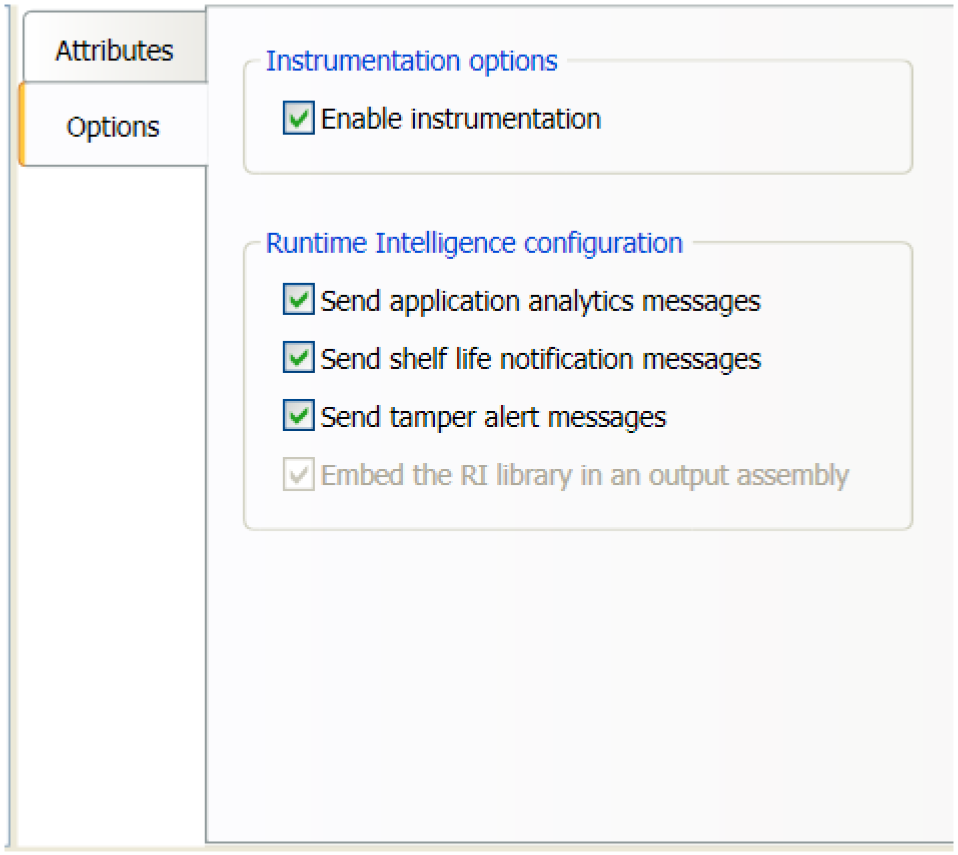
A Shelf Life Activation Key (SLAK) is required to instrument your application with this feature. To obtain a SLAK, please go to our Support site: **http://www.preemptive.com/support/index.html**.

InsertTamperCheckAttribute

The InsertTamperCheckAttribute injects code that detects if a user has modified your assemblies. If it detects that the assemblies have been altered, it can send a tamper notification message to the Runtime Intelligence Services Portal. Upon detection, the default behavior is to send the tamper notification message and exit the application; however, you may set the ApplicationNotificationSinkElement property to DefaultAction, which will cause the application to immediately exit. If you wish to send Runtime Intelligence tamper messages, do not put this attribute on the same method containing the Setup Attribute. Also, ensure that any methods marked with the tamper check attribute are executed after the method containing the setup attribute.

Instrumentation Options

The **Instrumentation Options** tab is where you may configure the behavior of your instrumented applications. Instrumentation (code injection) of your applications may be enabled or disabled for your current project by either un-checking the *Enable Instrumentation* check box, or by right clicking on Instrumentation in the navigation tree and un-checking the **Enable** option. If this option is unchecked, Dotfuscator will not perform code injection when your project is built.



Instrumentation options

✓ Enable instrumentation

Runtime Intelligence configuration

✓ Send application analytics messages

✓ Send shelf life notification messages

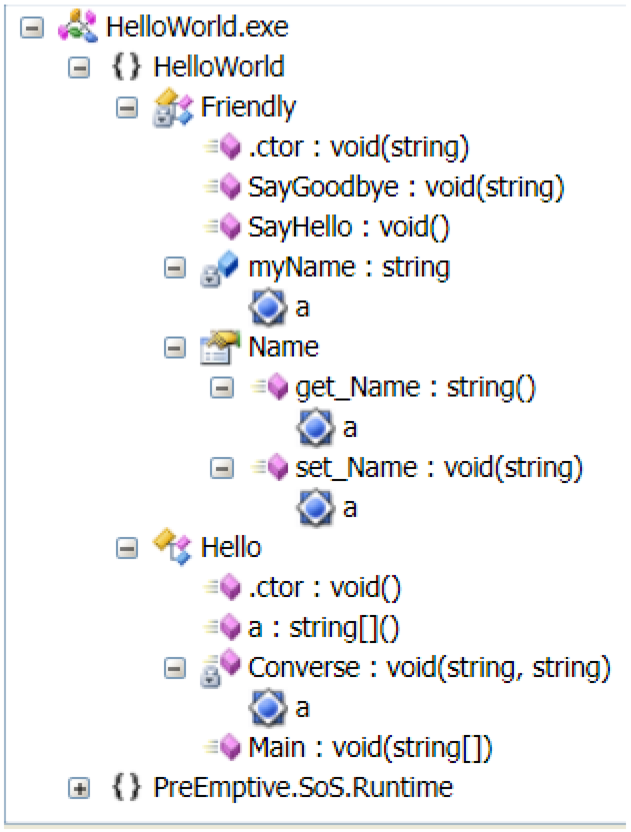
✓ Send tamper alert messages

✓ Embed the RI library in an output assembly

The *Runtime Intelligence configuration* section of this tab is where you may select which message types will be sent to the the Runtime Intelligence Services Portal. The Send Application Analytics messages option enables the sending of messages related to feature usage. The Send Shelf Life Notification messages option enables the sending of shelf life expiration and sign of life messages. The Send Tamper Alert messages option enables the sending of tamper notification messages. Un-checking either the Shelf Life or Tamper message options will only affect the transmission of those messages; if you have selected the DefaultAction to occur, your application will still exit as expected.

Results

Once your project is built, you can inspect the results in the *Results* Tab.



```
HelloWorld.exe
  { } HelloWorld
    Friendly
      .ctor : void(string)
      SayGoodbye : void(string)
      SayHello : void()
      myName : string
        a
      Name
        get_Name : string()
          a
        set_Name : void(string)
          a
    Hello
      .ctor : void()
      a : string[]()
      Converse : void(string, string)
        a
      Main : void(string[])
  { } PreEmptive.SoS.Runtime
```

Here you can browse the tree view and see how Dotfuscator renamed your types, methods, and fields. The new names appear as child nodes under the original nodes in the tree. This information is also saved to the map file as configured in the **Renaming Options**.

Beyond Obfuscation

Dotfuscator CE combines its trusted obfuscation technology with injection technology to provide near real-time views into application deployment, stability, and usage. In addition to obfuscation, Dotfuscator CE offers a new category of detective control to better monitor, manage, and protect your applications. Dotfuscator CE can:

- Provide a near real-time view into application integrity and activity at runtime.
- Instrument applications to detect if they have been tampered with and if so, optionally send a message to the Runtime Intelligence Service.
- Instrument applications with expiration, de-activation, and notification logic that reacts to application expiration by exiting the application and/or sending a Runtime Intelligence Services message.

In this section

Introduction to Active Tamper Defense

Dotfuscator CE provides a way for your applications to detect and optionally notify you if they have been tampered with since they were instrumented with tamper notification.

To detect tampering, place `InsertTamperCheck` attributes on one or more methods in the application that are always executed. When Dotfuscator encounters an `InsertTamperCheck` attribute during its processing, it adds code that performs an assembly level integrity check at runtime. If the integrity check fails, it sends a tamper detected message to the Runtime Intelligence service. It can also call code that exits the application. `InsertTamperCheck` attributes are not required at runtime; therefore, Dotfuscator strips them from the output application.

An application can contain any number of `InsertTamperCheck` attributes. In the event that an application has been tampered with, multiple tamper detected messages from the same application session will be sent with the same group ID.

Do not put this attribute on the same method containing the Setup Attribute. Methods with this attribute must be executed after the method containing the Setup Attribute.

In Dotfuscator CE, `InsertTamperCheck` attributes may only be placed in assemblies that are EXEs. Dotfuscator CE does not support tamper detection on library assemblies (DLLs).

In this section

Simulating Tampering

Supported .NET Application Types

Simulating Tampering

Dotfuscator ships with a simple command line utility that ‘tampers’ with an assembly. It is called **TamperTester.exe** and is installed in the same folder as Dotfuscator itself.

Usage

```
tampertester <file_name> [destination folder]
```

By running your Dotfuscator assemblies through this utility, you can test that the tamper notification messages are being generated and sent as expected. You can also test any application code you have written to execute in response to tamper detection.

Supported .NET Application Types

Dotfuscator CE can perform Tamper Notification processing for all .NET assemblies except for the following:

- Managed C++ input assemblies containing native and managed code.
- Multi-module input assemblies.
- Input assemblies that target .NET 1.0
- .NET Compact Framework assemblies.
- Silverlight assemblies.
- Any library assemblies.

Application Expiration

Shelf Life is an application inventory management function that allows you to embed expiration, or de-activation, and notification logic into an application via code injection. Dotfuscator injects code that reacts to application expiration by exiting the application and/or sending a Runtime Intelligence message. This functionality is helpful with beta applications, as it enables applications to self regulate and it enforces aging and expiry policies. You can schedule your application's expiration/de-activation for a specific date or a number of days after application instrumentation.

In this section

Shelf Life Activation Key Overview

Shelf Life Activation Key Overview

A Shelf Life Activation Key (**SLAK)** is a data file that is required to inject Shelf Life functionality into the appropriate locations within your application.

A Shelf Life Activation Key is issued by PreEmptive and provided to Dotfuscator by the user during shelf life configuration. To obtain a Shelf Life Activation Key, contact PreEmptive Solutions. PreEmptive will issue you a data file containing the Shelf Life Activation Key that is to be stored on your Build Machine.

Once you obtain a Shelf Life Activation Key, you can add the Shelf Life Attribute to a method or group of methods. In the *Instrumentation* tab in the *Attribute Editor*: section, in the *ActivationKeyFile* field you must select the path to the Shelf Life Activation Key file, thereby activating Shelf Life within your application.

Application Analytics

Runtime Intelligence is a service and a technology that gives application authors and users insight into how their applications are being used. Dotfuscator can be used to Runtime Intelligence Enable .NET applications and components.

Dotfuscator is used to instrument an application such that a message is sent when the application starts and stops and when designated features are being used. The Runtime Intelligence Service aggregates this lifecycle data from the application and exposes it through the Runtime Intelligence Portal.

In this section

Supported .NET Application Types

Supported .NET Application Types

Dotfuscator can perform Runtime Intelligence processing for all .NET assemblies except for the following:

- Managed C++ input assemblies containing native and managed code.
- Multi-module input assemblies.
- Input assemblies that target .NET 1.0

Free Runtime Intelligence Services Portal

Once your application has been instrumented, it will begin sending usage data to the Free Runtime Intelligence Services Portal where you may view a variety of reports generated from the accumulated data. The free Runtime Intelligence Services Portal provides for limited storage of usage data and has no minimum service level agreement and no user authentication; therefore, your data are considered public. For permanent data storage, additional summary and detail views of your data, and guaranteed SLAs, please contact us for an evaluation of our commercially available version.

The free Runtime Intelligence Services Portal provides the following reports:

- **Application Overview** - provides a high level, visual representation of the Operating Systems, environments, frameworks, and stability of your applications.
- **Application Scorecard** - provides high level textual representation of the usage of your applications broken down by the version information embedded in your applications.
- **Feature Overview** - provides a visual representation of the feature usage data gathered from your applications.
- **Feature Scorecard** - provides a high level textual representation of the feature usage gathered from your applications.

To begin viewing how your users are using your applications please go to http://free.runtimeintelligence.com and enter the **CompanyKey** that you used when instrumenting your application as the user name.

More detailed information about the reports is available from the Help topics located on the free Runtime Intelligence Services Portal.

Index