Dotfuscator Software Services Instrumentation Tutorial

Application Protection and Analytics

“We view PreEmptive Solutions' instrumentation platform for commercial software development as game-changing."

– Director of Architecture, TMW Systems

Table of Contents

Dotfuscator Software Services Instrumentation Tutorial

Dotfuscator is more than a .NET Obfuscator and Compactor. It is also the platform on which applications can be instrumented for detailed, granular analytics and reporting via Runtime Intelligence Services. By using Dotfuscator to instrument applications, users can have invaluable insight into how their applications are being used, who their users are, and how well their applications are running.

This tutorial will show you how to instrument your applications so that you can take advantage of all the solutions using Runtime Intelligence Services will provide you.

Getting Started

Explains how to instrument Dotfuscator so that application analytics and messages can be sent to Runtime Intelligence Services.

Application Hardening and Security

Describes the benefits of application hardening and security and the steps to take to ensure your application is safe from reverse engineering.

Application Stability and Feature Usage

Describes how to instrument your application to determine stability on various frameworks and operating systems and to determine feature usage.

Application Analytics

Explains how to instrument an application for application analytics as well as the reports generated from instrumentation.

Customization

Straightforward explanation of integrating instrumentation with custom code.

Getting Started

Before you begin the tutorial there are several assumptions this tutorials makes and several requirements that must be met:

Assumptions

This document assumes familiarity with Visual Studio, Dotfuscator, and obfuscation (using Dotfuscator).

Requirements

Users must have the following:

- Visual Studio 2005 or higher
- Dotfuscator Pro*
- MedicalImage

In order to follow this document, the tutorial application, MedicalImage, requires version 3.0 or later of the .NET runtime, and is available from the Microsoft Download Center at **http://www.microsoft.com/downloads/details.aspx?familyid=3026A2D3-E48B-4F5E-8B5F-03AF30D2486F&displaylang=en**.

- In addition to having Dotfuscator Professional, the Shelf Life and Sign of Life portions of the tutorial require that you have a **Shelf Life Activation Key (SLAK)** file issued by PreEmptive Solutions.

*Users of Dotfuscator Community Edition can still follow this tutorial, however, those functions that CE does not have access to will be disabled in the User Interface.

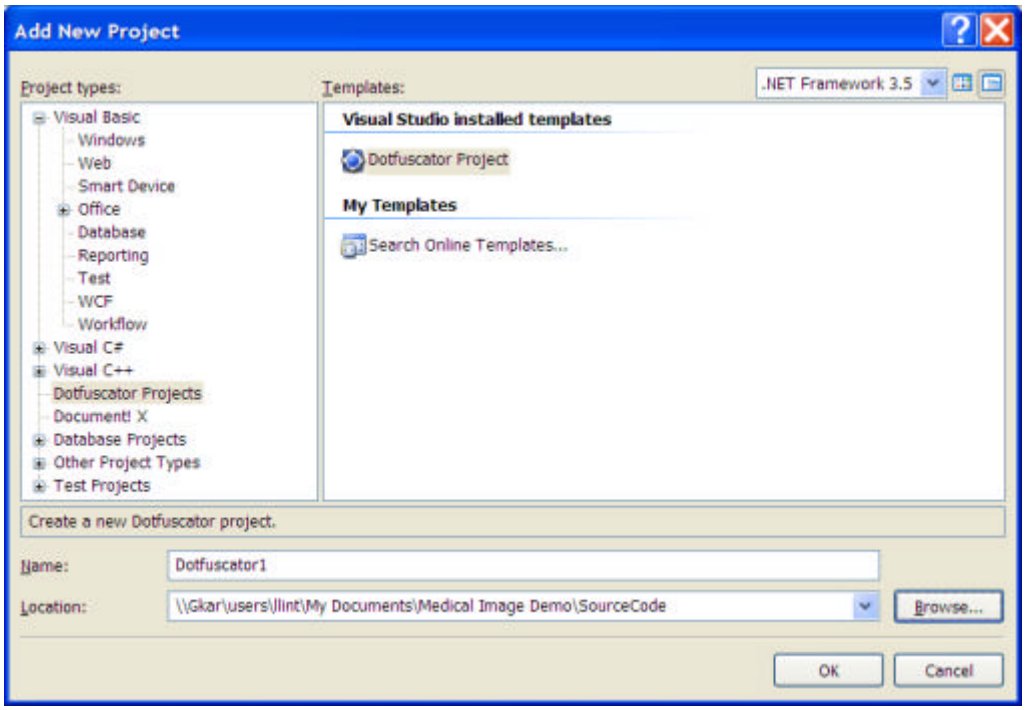
In this section

Setting up the Project

Setting up the Project

To to begin instrumenting an application, you must set up the project to be instrumented. Begin by performing the following steps:

1. As mentioned in **Getting Started**, download the MedicalImage project from the Microsoft Download Center.
2. Launch Visual Studio. Once Visual Studio is opened, click **File > Open > Project/Solution**.
3. The Open File window displays. Select the the MedicalImage solution file (**MedicalImage.sln**), then click **OK**.
4. In the Solution Explorer pane, right click the solution and select **Add > New Project...** The **Add New Project** window displays.
5. Select **Dotfuscator Projects** from *Project Types*:, and then select Dotfuscator Project as your project as shown here:

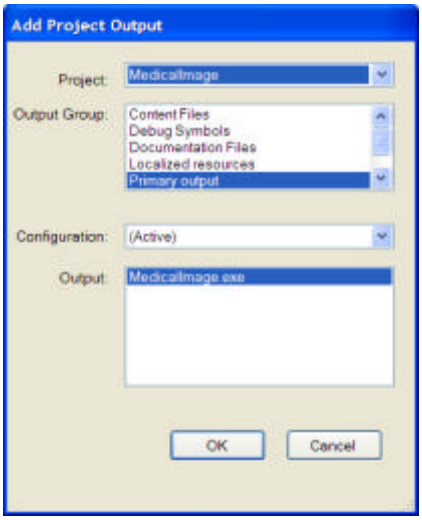


Note: The *Name:* field is automatically populated with the default project name **Dotfuscator1**.

Routing the Output

Once the original MedicalImage and the Dotfuscator projects are set up, you must route the output from the MedicalImage project build to the input for Dotfuscator by performing the following steps:

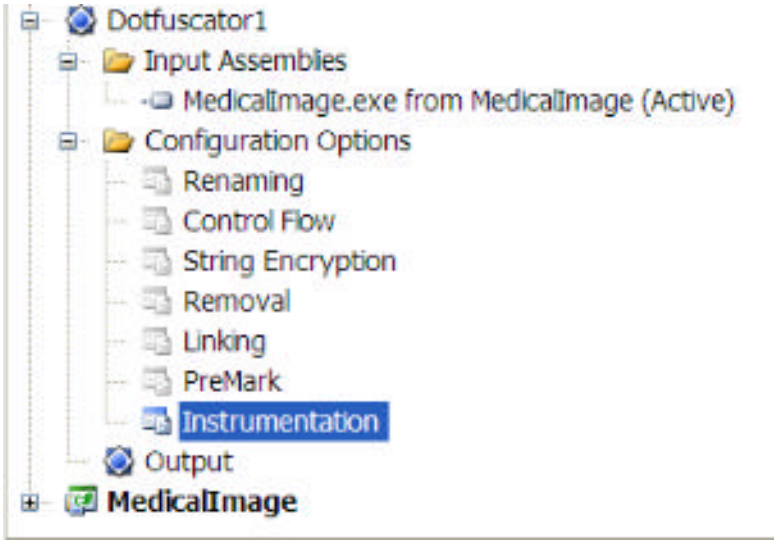
1. Right click the **Input Assemblies** icon in the *Solution Explorer* pane under the Dotfuscator project and select **Add Project Output...**.
2. The **Add Project Output** window displays.
    1. In the *Project:* field, select the **MedicalImage** prject.
    2. In the *Output Group:* field, select **Primary output**.
3. Click **OK**.



Now you are ready to prepare the application for instrumentation.

Preparing the Application

In order for your application to send analytics messages to Runtime Intelligence Services, the application must be uniquely identifiable. This is accomplished by performing the following steps:

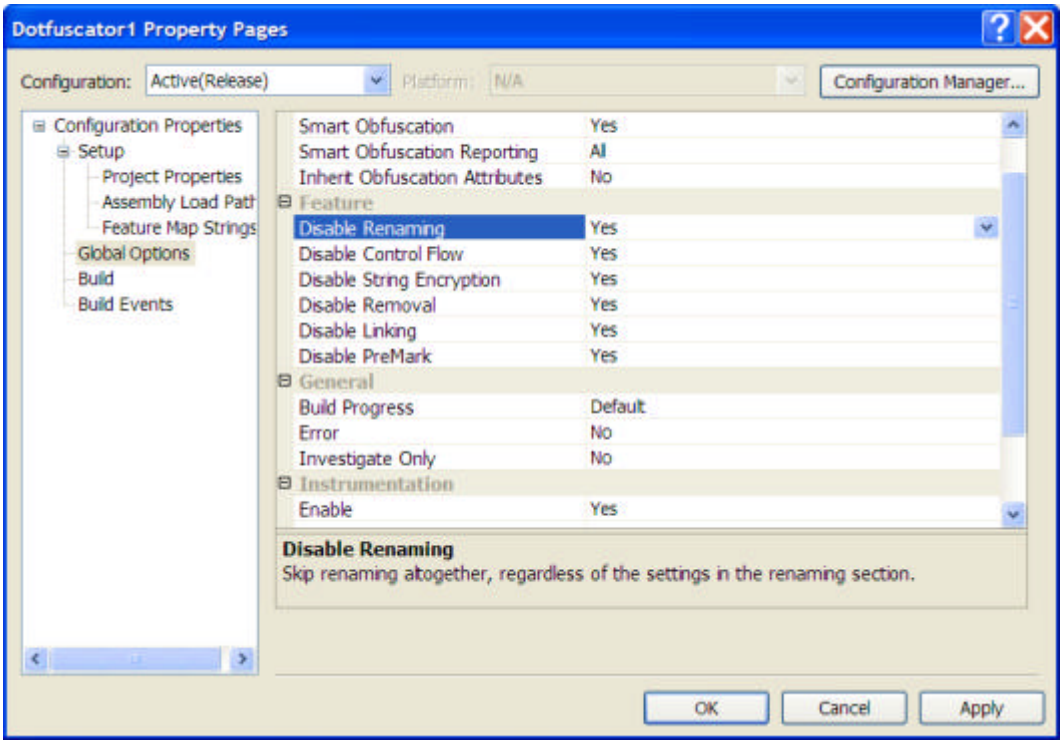


1. In the Dotfuscator project, select **Instrumentation** in the Solution Explorer to open the Instrumentation tab.
2. When the Instrumentation tab opens, right click on the top level node and select **Add Attribute** from the context menu.
3. The **Add Attribute** window displays. Click on and select **PreEmptive.Attributes.BusinessAttribute** and then click **OK**.
4. Within the *Attribute Editor* in the *CompanyKey* field, enter the **key** you receive either as a stand-alone Runtime Intelligence subscription or bundled into Dotfuscator Professional. Enter the **name** you wish to be reported as your company name in the *CompanyName* field.
5. Right click the top node again so that you may add an **ApplicationAttribute** which is used to provide unique identifying information to the Runtime Intelligence service to report on multiple versions of your application. Each application should have a unique *GUID* and *Name* value per application with successive versions having a different *Version* value. The information entered into this attribute is relayed to the Runtime Intelligence reporting service.
6. Optionally add a **BinaryAttribute** which enables Runtime Intelligence to identify which assembly within the application has been tampered with.

The GUIDs for the Binary and Application attributes can be created from within the attribute properties by clicking on the "…" on the GUID property, from the **Create GUID** option on the Visual Studio Tools menu, or by a tool such as guidgen.exe. These GUIDs identify your application and assemblies to Runtime Intelligence and should stay constant through the life of your application.

Since the objective of this tutorial is to provide you with an understanding of instrumentation and it is assumed that you are familiar with obfuscation, Renaming, Control Flow, and String Encryption can be disabled. To disable these features, right click on the project in the Solution Explorer and select **Properties**.

When the Property Pages window displays, click on **Global Options**.



In the *Feature* section of Global Options, set **Disable Renaming**, **Disable String Encryption**, and **Disable Control Flow** to '**Yes**,' click **Apply**, then click **OK**.

You are now ready to begin instrumenting your application for **hardening and security**, **stability monitoring**, and **feature usage tracking**.

Application Hardening and Security

Programs written for .NET are easy to reverse engineer. Attackers can use a .NET decompiler to easily reverse engineer code, exposing copy protection mechanisms and proprietary business logic - whether it's legal or not. Organizations concerned with their intellectual property on the .NET platform must harden and secure their applications.

While obfuscation may deter attackers from reverse engineering your application and stealing your source code, **Tamper Detection and Notification** detects and notifies you of such attempts, including who was tampering with your application and when they were doing so. Adding **Shelf Life** to your applications enforces expiration policies and limits the duration of time the application is functional.

This section of the Tutorial explains how to add the Tamper Detection, Shelf Life, and Sign of Life attributes and notification/messaging logic to an application.

In this section

Tamper Detection and Notification

Shelf Life and Sign of Life

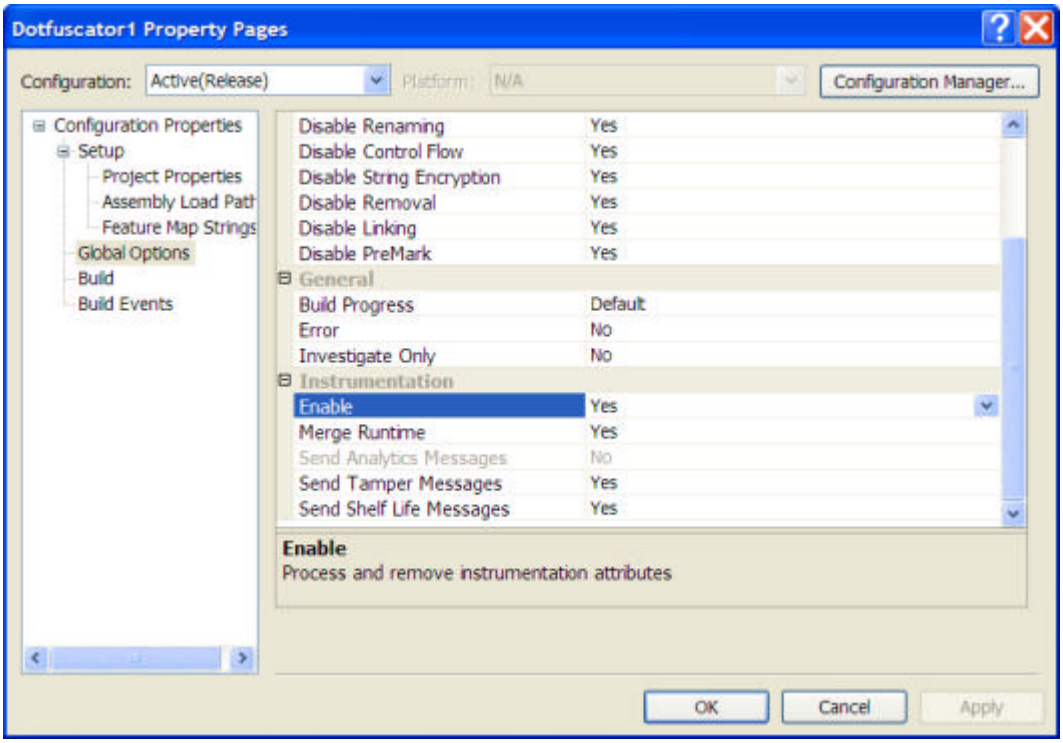
Tamper Detection

Dotfuscator can instrument applications to detect if tampering has occurred. For example, if an attacker attempts to circumvent your application's licensing requirements or modify your binary, you would not know about it unless your application was instrumented with tamper detection and notification. In essence, *Tamper Detection* detects unauthorized tampering of your application's source code. Once the tamper is detected you can receive notification, shut down the application, or take other customized actions.

Instrumenting the Application

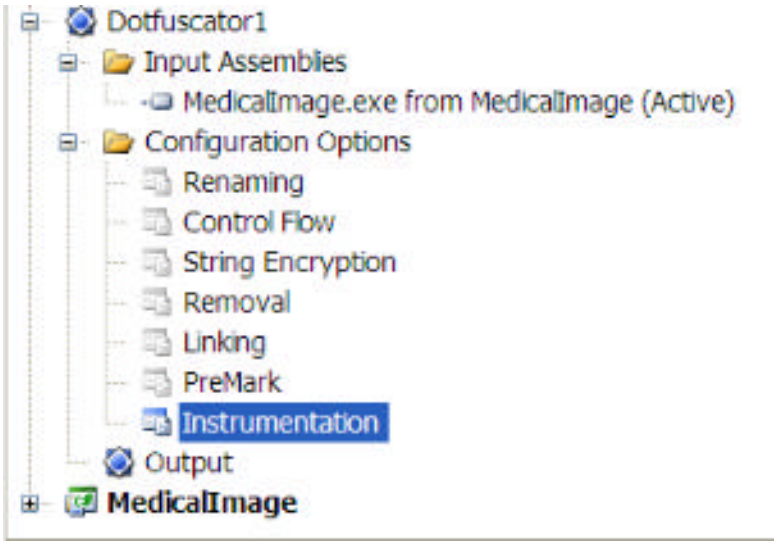
You are now ready to instrument your application for **hardening and security**, **stability monitoring**, and **feature usage tracking**. Perform the following steps to begin instrumenting your application:

1. Right click the Dotfuscator project in Solution Explorer and select **Properties** from the context menu.
2. The **Project Property Pages** window displays. Click on **Global Options**.
3. In the **Instrumentation** section of the grid, set **Enable** to '**Yes**'. Selecting **Yes** allows the application to be instrumented.
4. Click **OK**.

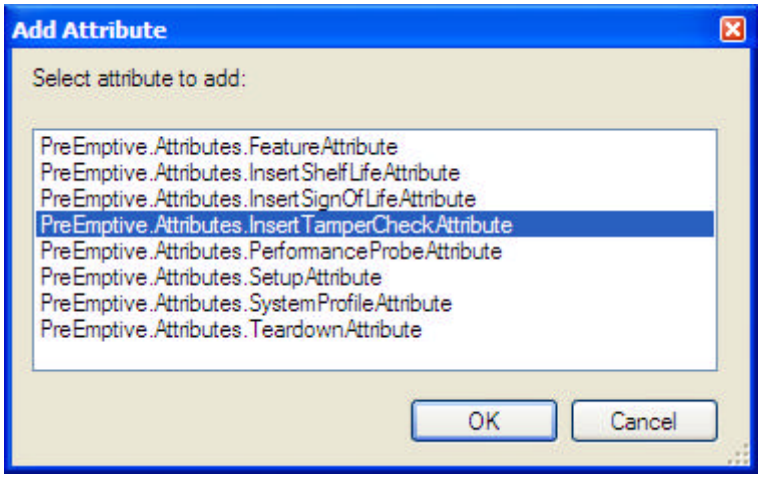


A tamper check occurs on any method instrumented with an **InsertTamperCheck** attribute. This attribute enables Dotfuscator to inject the tamper detection, defense, and notification routines at the appropriate locations within an application. For our example, we will instrument the **InitializedComponents** method.

To add the **InsertTamperCheck** attribute, do the following:

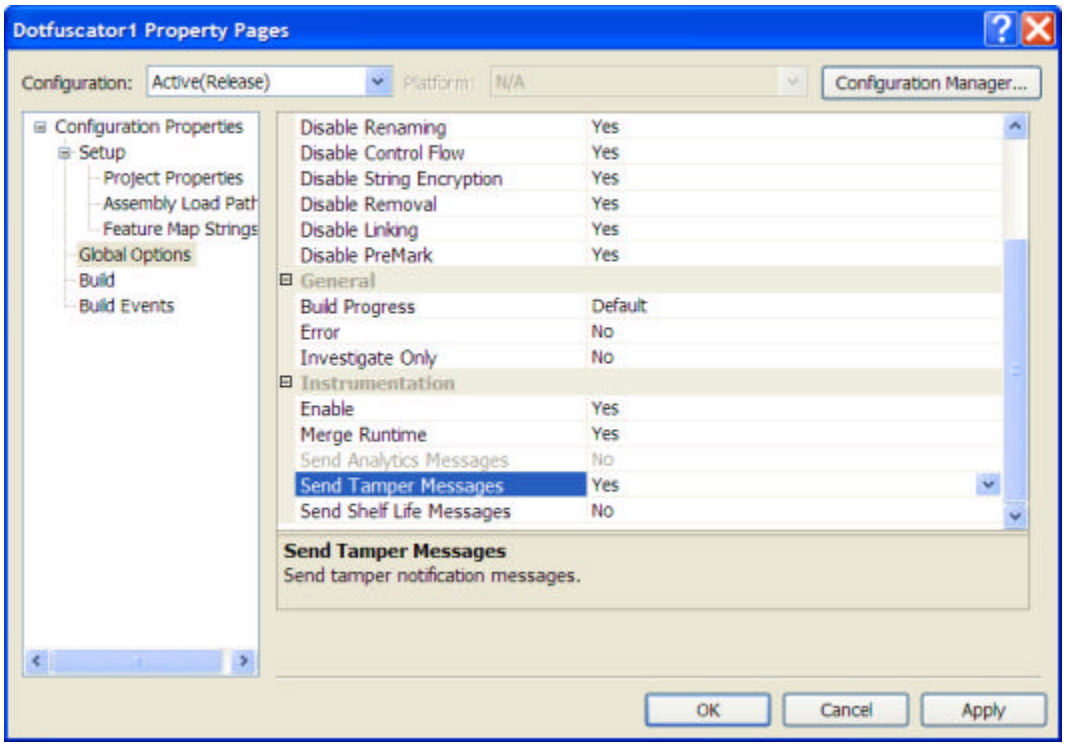


1. Open **Instrumentation** on the Dotfuscator project in the Solution Explorer.
2. Locate the **InitializeComponent** method in the **Window1** class.
3. Right click the **InitializeComponent** method and select **AddAttribute**.
4. The **AddAttribute** window displays. Select **InsertTamperCheck** from the list of available attributes.
5. Click **OK**.



The application is now instrumented with Tamper Detection.

Next, you can instrument the application to send tamper notification messages:



1. Right click the Dotfuscator project in Solution Explorer and select **Properties** from the context menu.
2. The **Project Property Pages** window displays. Click on **Global Options**.
3. In the **Instrumentation** section of the grid, set **Send Tamper Messages** to '**Yes**'. Selecting **Yes** allows tamper notification messages to be sent to Runtime Intelligence Services which in turn sends an email notification of the tamper occurrence.
4. Click **OK**.

Your next steps are **building** the application instrumented with Tamper Detection and Notification and **testing** it to ensure that a Tamper Notification message is received.

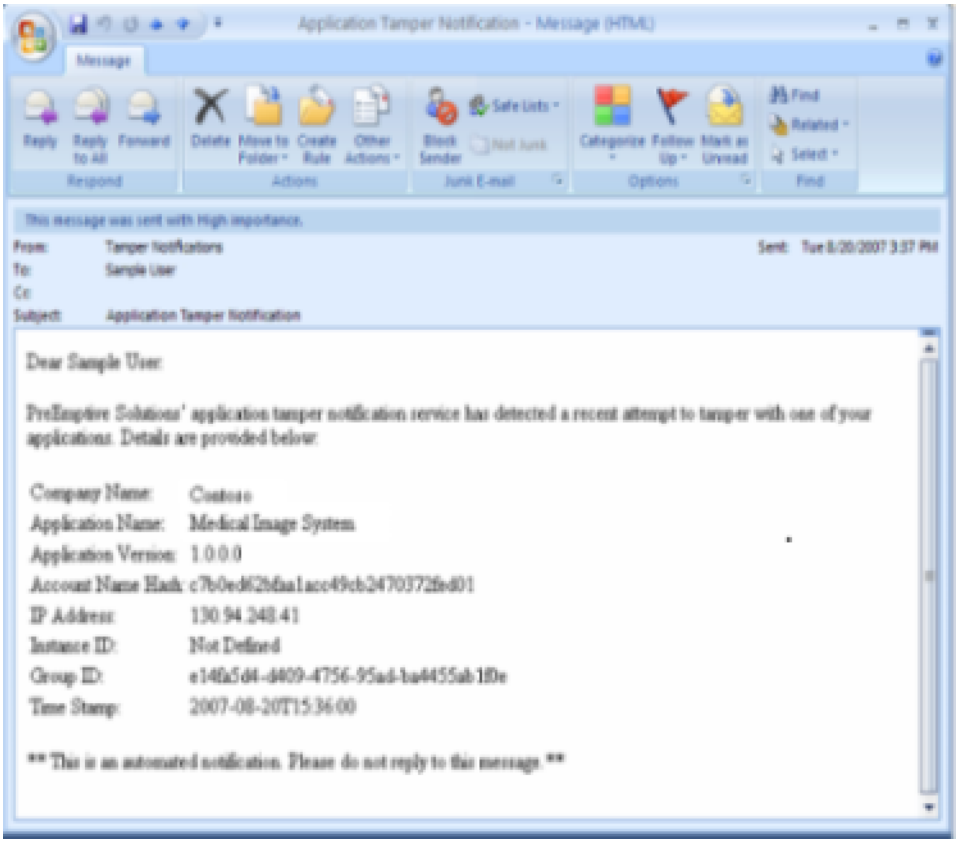
Building and Testing a Tampered Application

Now that the MedicalImage application is instrumented with Tamper Detection and Notification, it must be tested to ensure that every instance of the application will automatically detect any tampering and transmit a secure SSL signal that results in a near real-time incident report delivered to your organization with links to additional forensic data.

Dotfuscator includes a tamper testing tool that developers can use to verify functionality. This is a command line tool that is installed in the Dotfuscator folder (typically at **C:\Program Files\PreEmptive Solutions\Dotfuscator Professional Edition 4.4\TamperTester.exe**).

To test the application:

1. Run Dotfuscator by right clicking the Dotfuscator project and selecting **Build**. The output window will show that the MedicalImage builds first, followed by Dotfuscator's post-build processes that are governed by the options that you have set.
2. Once it is built, copy the MDB database file to the Dotfuscator output directory.
3. From the Dotfuscator output directory, run **tampertester MedicalImage.exe**.
4. Tampertester will create a modified, but usable, version of MedicalImage.exe in a subfolder called **Tampered**.
5. Copy the MDB file to this directory.
6. Run the tampered application.
7. After some time, you should receive a tamper notification such as this:



Receiving this message verifies that the application has been instrumented with Tamper Detection and Notification.

Shelf Life and Sign of Life

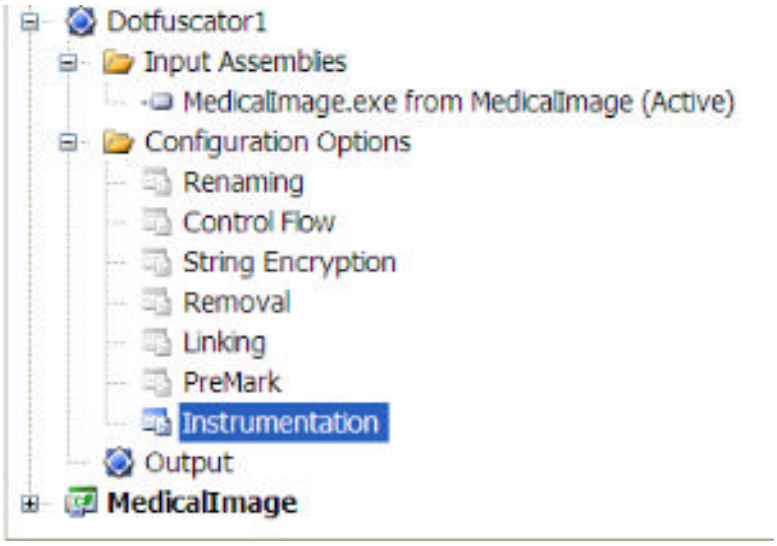
Dotfuscator can be used to instrument applications to detect *when* they are being used and to *limit* the duration of time those applications can be used. In addition, those applications instrumented with an expiration date can also be instrumented to send a message warning of an impending expiration as well as a message stating expiration has occurred. For example, if you provided a client with a Beta Release or Evaluation copy of your software with the stipulation that they can use it for 14 days from the date of build only, you would not know if your application was downloaded, installed, or even used unless it was instrumented with Shelf Life and Sign of Life. In essence:

- Sign of Life detects the heartbeat of your application;
- Sign of Life Notification is a message that you can optionally receive informing you your application has been used.
- Shelf Life enforces expiration policies ensuring that your application is available for use in the time allotted only;
- Shelf Life Notification is the message you can optionally receive informing you of expirations that occurred or warning you of an impending expiration.

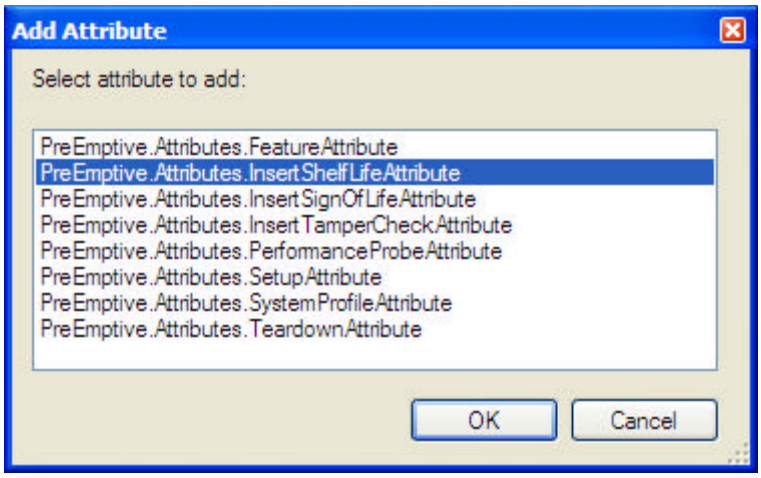
Instrumenting the Application with Shelf Life

Note: The Shelf Life functionality requires a Shelf Life Activation Key (SLAK) provided by PreEmptive Solutions. You must first obtain a **Shelf Life Activation Key (SLAK)** to add this attribute to an application. The key is issued by PreEmptive and provided to Dotfuscator by the user during shelf life configuration and is required at the time of obfuscation.

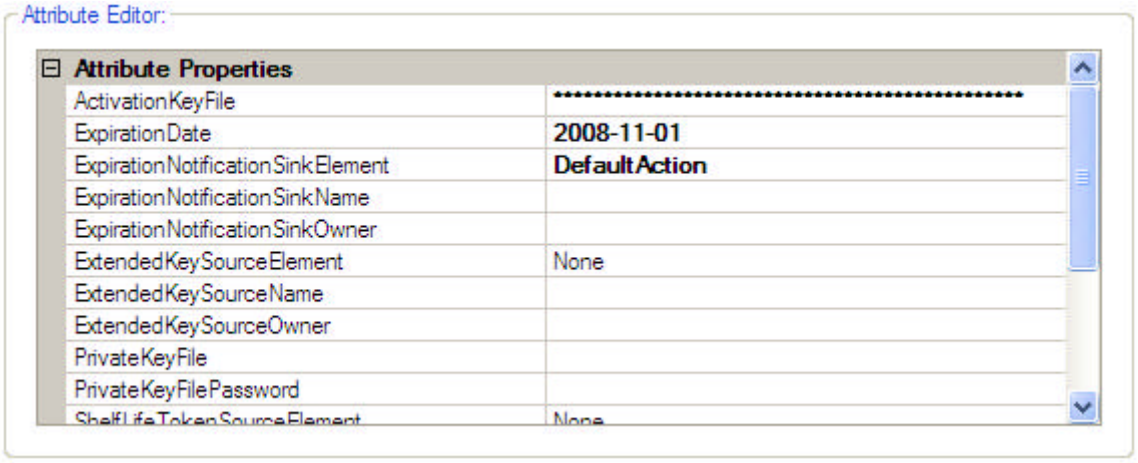
You are now ready to instrument your application for Shelf Life and Sign of Life. Perform the following steps to begin instrumenting your application:



1. Open **Instrumentation** on the Dotfuscator project in the Solution Explorer.
2. Locate the **InitializeComponent** method in **Window1** class.
3. Right click the **InitializeComponent** method and select **Add Attribute**.
4. The Add Attribute window displays. Select **InsertShelfLife** attribute from the list of available attributes.
5. Click **OK**.

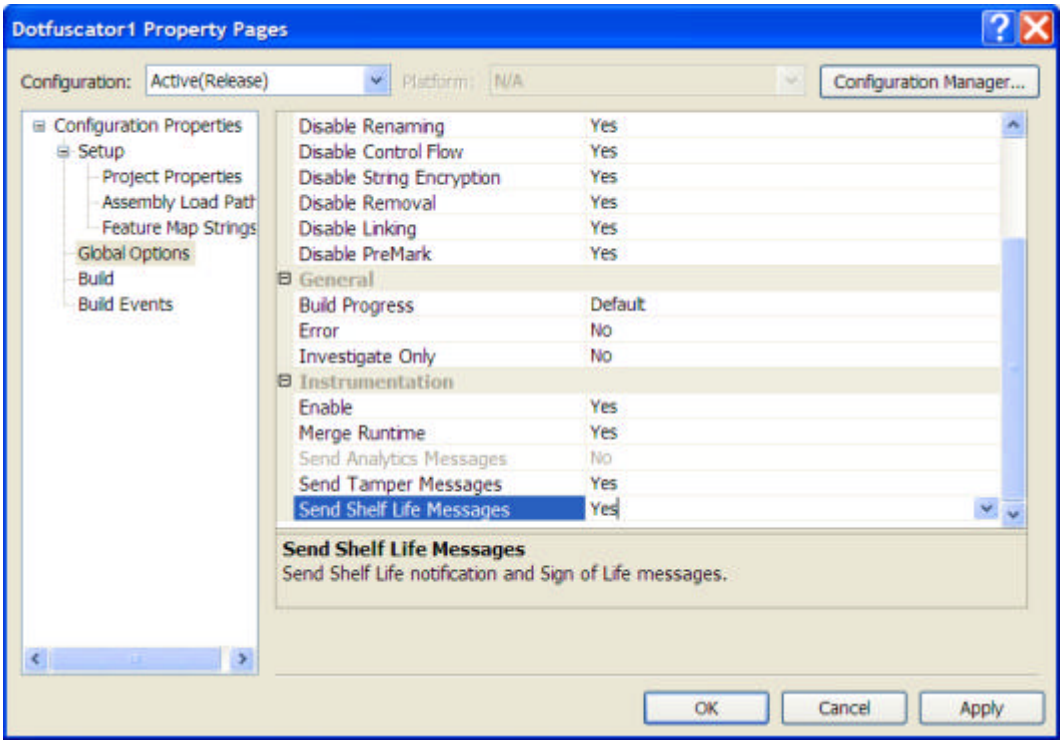


The **InsertShelfLife** attribute indicates where in the application the test for expiration should occur. This can be included on multiple methods.



6. In the *Attributes Editor:* section, enter the **file path** to the location of the **Shelf Life Activation Key** in the *ActivationKeyFile* field.
7. Click the drop down arrow in the *ExpirationDate* field and select the **date** the application is to deactivate or enter the number of days from the date of build that the application is to deactivate.
8. Click the drop down arrow in the *ExpirationNotificationSinkElement* field and select **DefaultAction**. By selecting **DefaultAction**, you are instructing Dotfuscator to inject code that exits the application if it is expired.
9. Click the drop down arrow in the *WarningDate* field and select the **date** the application is to begin warning of an impending deactivation or enter the number of days from the date of build that the application is to notify of an impending deactivation.
10. You can test the Shelf Life expiration behavior by setting the *Expiration Date* on the Shelf Life attribute to a date in the past. Run the instrumented application, which should then immediately terminate.

The application is now instrumented with Shelf Life. Next, you can instrument the application to send Shelf Life Expiration and/or Warning messages.



1. Right click the Dotfuscator project in Solution Explorer and select **Properties** from the context menu
2. The **Project Property Pages** window displays. Click on Global Options.
3. In the Instrumentation section of the grid, set **Send Shelf Life Messages** to '**Yes**'. Selecting **Yes** allows Shelf Life expiration and warning messages to be sent to Runtime Intelligence Services of the expired application.
4. Click **Apply**, then click **OK**.
5. Repeat steps.

To test the notification, set the *Warning* and/or *Expiration* dates in the Shelf Life attribute to a date in the past, rebuild the solution, and run the obfuscated and instrumented application. You will then be able to view any Shelf Life warning or expiration occurrences in the Runtime Intelligence portal after they have been processed by the Runtime Intelligence service.

Instrumenting the Application with Sign of Life

Sign of Life is a feature that enables application owners to track when their applications instrumented with Shelf Life are started, and how frequently those applications are started.

To Instrument your application with Sign of Life, do the following:

1. Open **Instrumentation** on the Dotfuscator project in the Solution Explorer.
2. Locate the **InitializeComponent** method in **Window1** class.
3. Right click the **InitializeComponent** method and select **Add Attribute**.
4. The Add Attribute window displays. Select **InsertSignofLife** attribute from the list of available attributes.
5. Click **OK**.

Application Stability and Feature Usage

In addition to hardening and securing your application with **Tamper Detection** and **Shelf Life and Sign of Life**, Dotfuscator can be used to instrument your application with Runtime Intelligence to gain unprecedented near-time views into:

- Application Stability across domains and platforms
- User behavior
- Feature Usage and Tracking

In other words, you can instrument your application to discover on which platforms and operating systems your application functions well and on which it does not. You can also instrument your application to ascertain which features of your application are used most often, which features need to be updated, and those that can be phased out. Having insight into your application with data at your fingertips enables you to focus your development and marketing resources and efforts in areas where they can be most effective.

In this section

Application Stability

Feature Usage Tracking

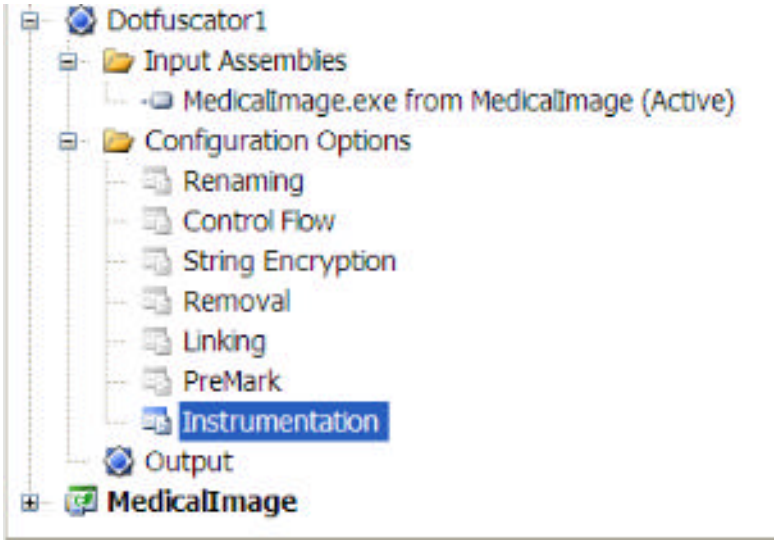
Application Stability

Dotfuscator can instrument applications to detect application stability. That is, it can detect when an application session was started and when it was shutdown or exited properly. In addition, your application can be instrumented to detect when an application was shutdown abnormally, for example, if an application crashes or the computer on which it is running experiences technical difficulties.

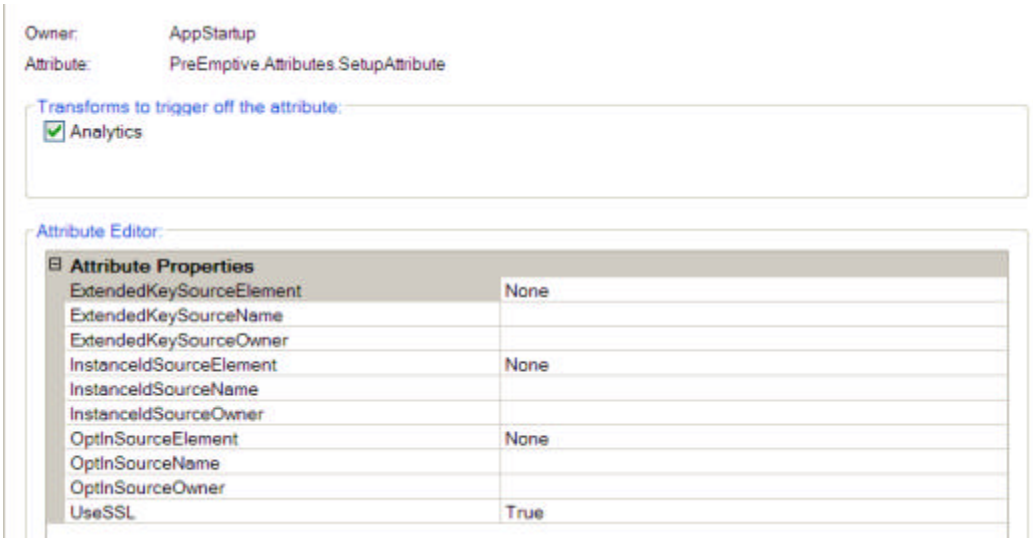
Setup (the application has started running) and **Teardown** (the application has stopped running) attributes need to be added to the application to track application runs.

Instrumenting the Application

To instrument the application for application stability monitoring and to add the Setup Attribute, perform the follow steps:



1. In the Dotfuscator project, select **Instrumentation** in the Solution Explorer to open the Instrumentation tab.
2. When the **Instrumentation** tab opens, expand the top level node.
3. Expand the following nodes: **MedicalImage.exe** > **{}MedicalImage** > **app**.
4. Right click on **AppStartup** (which is under **app**) and select **Add Attribute** from the context menu.
5. The **Add Attribute** window displays. Click on and select **PreEmptive.Attributes.SetupAttribute** and then click **OK**.
6. Click in and check the *Analytics* checkbox. By selecting Analytics you are enabling Dotfuscator to send an Application Start message to Runtime Intelligence Services.



Next, the Teardown attribute is to be added.

1. Expand the following nodes: **MedicalImage.exe** > **{}MedicalImage** > **app**.
2. Right click on **Main** (which is under **app**) and select **Add Attribute** from the context menu.
3. The **Add Attribute** window displays. Click on and select **PreEmptive.Attributes.TeardownAttribute** and then click **OK**.
4. Click in and check the *Analytics* checkbox. By selecting Analytics you are enabling Dotfuscator to send an Application Stop message to Runtime Intelligence Services.

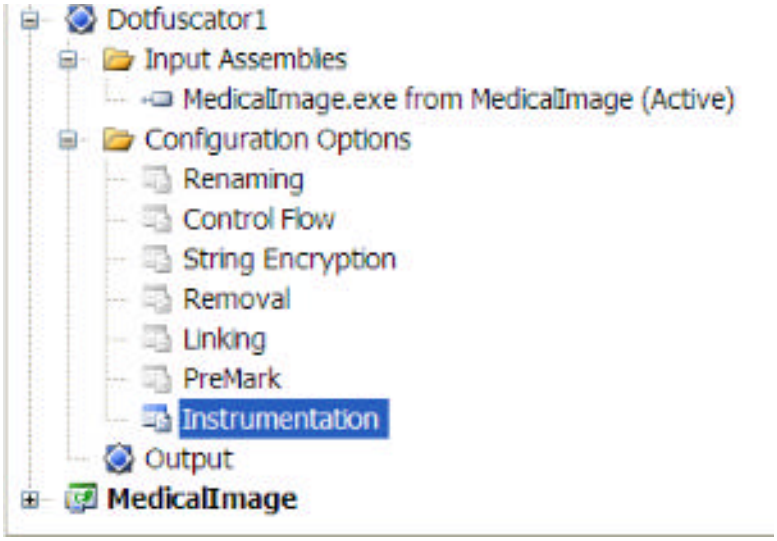
The information that is tracked here will display on the Runtime Intelligence Portal in the **Application Overview** reports.

Feature Usage Tracking

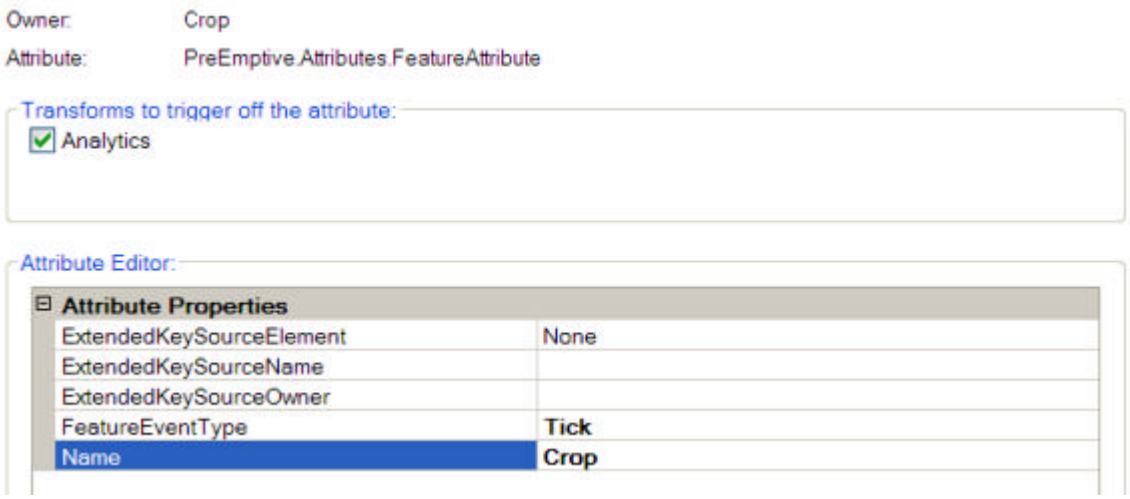
Dotfuscator can instrument applications to detect feature usage and duration of use. For example, by instrumenting all the features of your application, you can discover which features are used most frequently, which features may require updating, and which features can be phased out.

Instrumenting the Application

To instrument the application for feature tracking you will need to add the Feature Attribute. Perform the follow steps:



1. Open **Instrumentation** on the Dotfuscator project in the Solution Explorer.
2. Locate the **Crop()** method in the **Window1** class.
3. Right click the **Crop()** method and select **AddAttribute**.
4. The **AddAttribute** window displays. Select **InsertFeatureAttribute** from the list of available attributes.
5. Click **OK**.
6. Name the **FeatureAttribute** by setting its *Name* property to **Crop**.
7. Click in and check the *Analytics* checkbox. By selecting Analytics you are enabling Dotfuscator to send a Feature Usage message to Runtime Intelligence Services.
8. In the *FeatureEventType* field, select **Tick** from the drop down. Selecting **Tick** indicates feature usage. This will send a message to RIS each time the Crop feature is used.
9. Locate the **Load From Server** method in the **Window1** class again, then repeat steps 2-7.
10. In the *FeatureEventType* field, select **Start** from the drop down. Selecting **Start** indicates that the feature started. This will send a message to RIS each time the Crop feature is invoked.
11. Locate the **Load From Server** method in the **Window1** class again, then repeat steps 2-7.
12. In the *FeatureEventType* field, select **Stop** from the drop down. Selecting **Stop** indicates that the feature has completed. This will send a message to RIS each time the Crop feature has completed its function.



The report that results from the information gathered from an instrumented feature displays on the Runtime Intelligence Portal in the **Feature Overview Reports**. The *Tick*, or each time the feature was used, will display in the **Feature Usage Over Time** section of Overview. Each of the *Start* and *Stop* messages for the feature, that is when the feature was invoked and successfully completed what it was required to do, will be displayed in the **Feature Ratio for Uses** section of the Overview. The Incomplete Sessions of the Overview displays those times the feature was invoked, but a Stop message was not received. Such events occur, for example, if the computer on which the application was running crashed.

Application Analytics

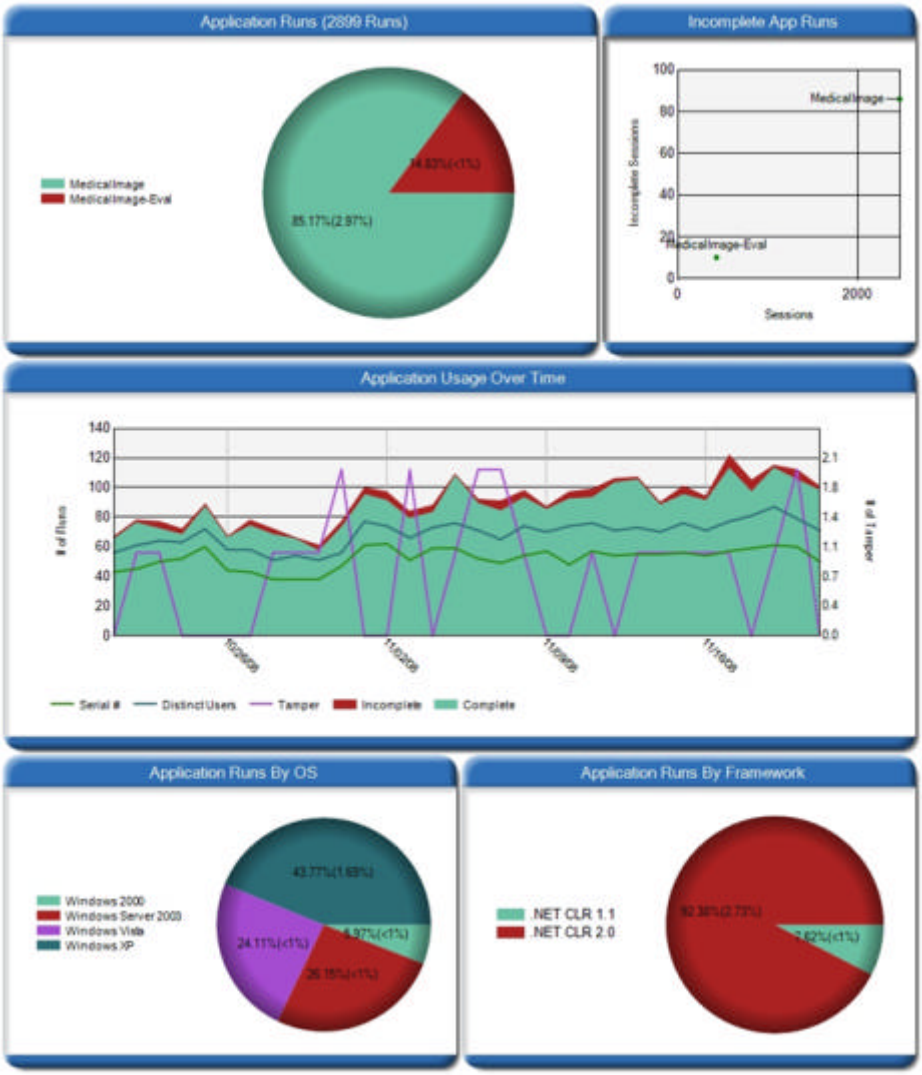
In the **Application Hardening and Security** section of the tutorial, you were shown how to instrument your application for Tamper Detection, Shelf Life expiration, and Sign of Life and to send notification messages when such events occurred. In the **Application Stability and Feature Usage** section, you were shown how to instrument your application to track feature usage and application stability and receive analytics, or data, on this information.

Once an application is instrumented and begins sending messages to Runtime Intelligence Services, Runtime Intelligence aggregates the data and presents it in graphs and charts on the Runtime Intelligence Services portal, which can be accessed at http://www.runtimeintelligence.com.

Note: Runtime Intelligence must be activated from a registered copy of Dotfuscator. For more information about the Runtime Intelligence Portal, please see the Runtime Intelligence Services Portal Users' Guide.

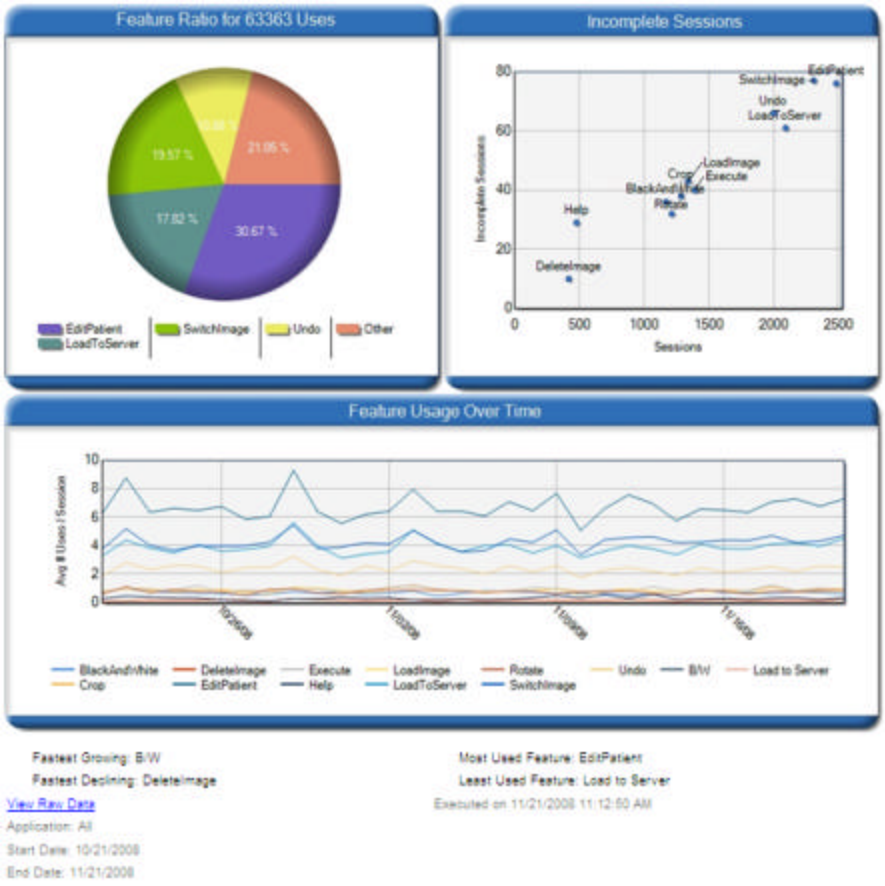
The Application Overview Reports

The initial application analytics dashboards offer insight into application adoption, usage, and stability in the Application Runs and Incomplete Runs portions of the Application Overview Reports page. Any Tamper Messages that were received are displayed in the Application Usage Over Time chart:



The Feature Overview Reports

Feature data are generated by tracking feature-level usage and behavior. The Feature Overview Reports display the most used features and it includes incomplete feature usage - instances where the application crashed or was shut down improperly while the feature was invoked. This report succinctly summarizes the fastest growing feature, the fastest declining feature, and the most and least used features at the bottom of the page.



Customization

Most commercial-grade applications require custom defense against tampering and application-specific registration and activation dialogs. Dotfuscator Professional's instrumentation is architected to integrate with custom code. See **Custom Tamper Dialog** and **Custom Shelf Life Behavior** for examples that illustrate the straightforward nature of this integration.

Custom Tamper Dialog

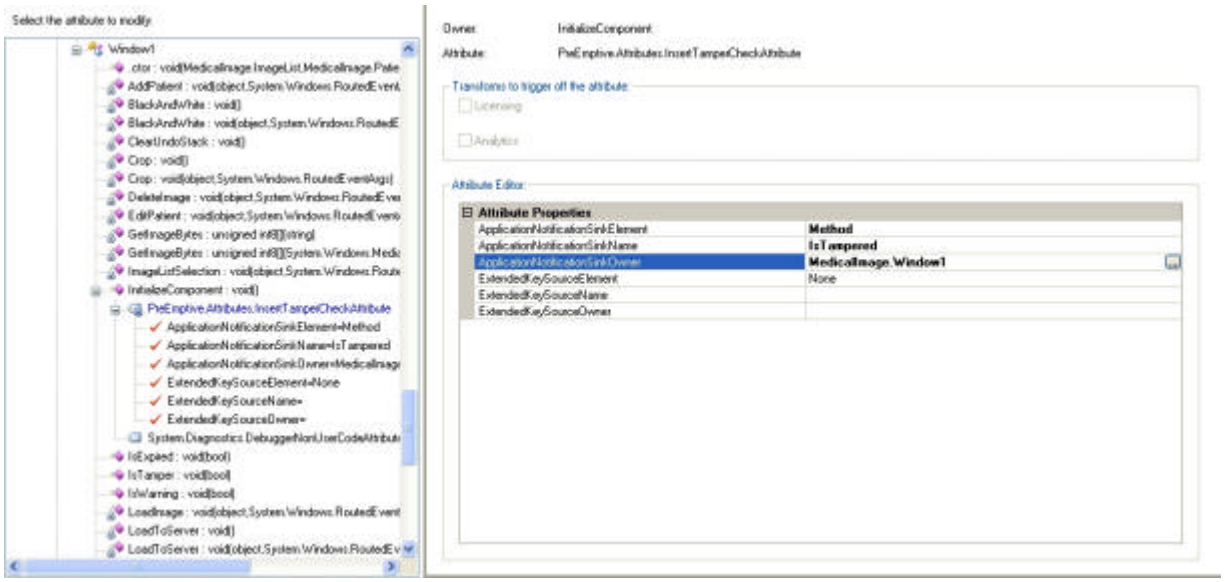
The following steps are all that are required to include a real-time, custom defense against tampering:

1. Open the project. **File > Open > Project/Solution** - select **Project**.
2. Verify that the **InsertTamperCheck** attribute added previously in this guide is added, or follow the prior steps to insert it.
3. Add the tamper-handling method to the Window1 class (defined in Window1.xaml.cs). A tamper handling method must take on Boolean parameter which will be set to True if tampering has been detected or False otherwise.

   Here is an example that shows a message and exits the application:

```
private void IsTampered(bool isTampered) {
    if(isTampered) {
    MessageBox.Show("Tampering Detected.  Shutting down...");
    System.Environment.Exit(0);
    }
}
```

4. Set the sink element type (*ApplicationNotificationSinkElement* property) to **Method**.
5. Set the *ApplicationNotificationSinkName* property to the name of the method to be called during the tamper check. In our example above **(isTampered)** is the name of the method to be called.
6. Ensure that *Instrumentation* is enabled in the Dotfuscator project's Global Options.
7. Build the solution. This will recompile the application and run Dotfuscator on the output.
8. As before, you can test using the Tamper Tester tool that comes with Dotfuscator.



Custom Shelf Life Behavior

The following steps are required to include custom actions that can be taken upon application expiration or entry into the expiration warning timeframe.

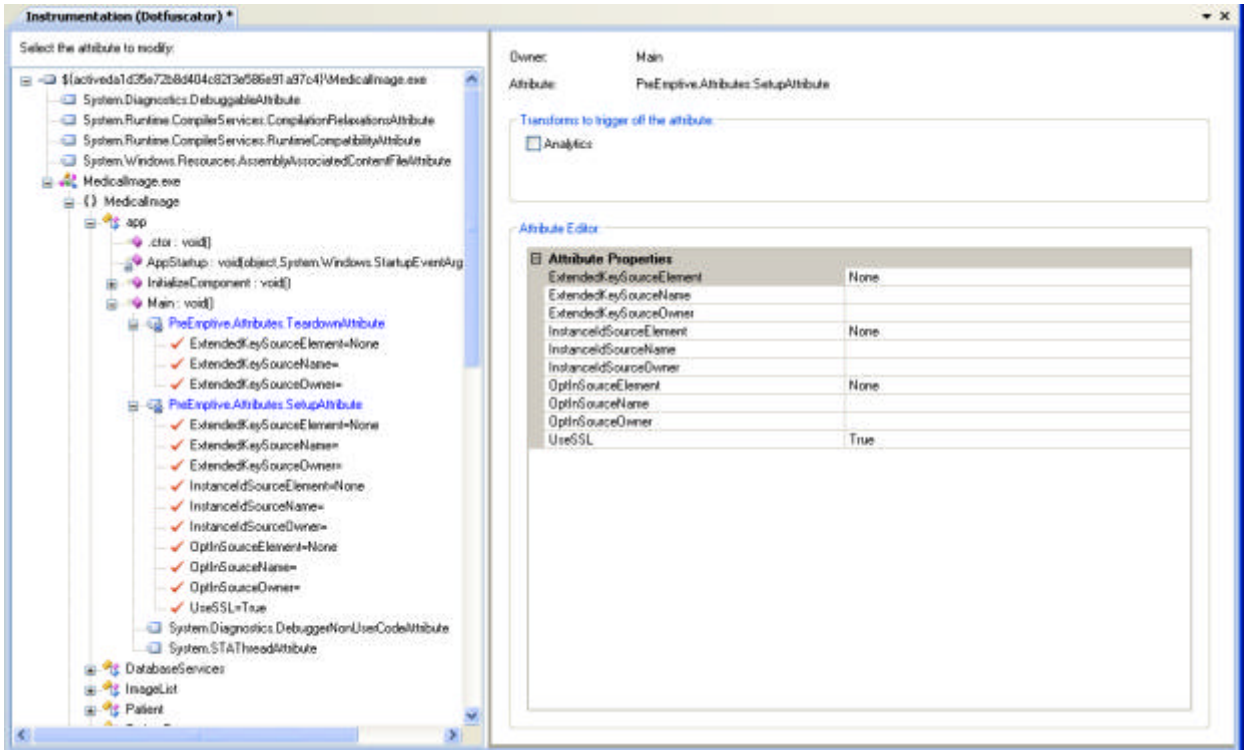
1. Open the project. **File > Open > Project/Solution** - select **project**.
2. Verify that the InsertShelfLife attribute added previously in this guide is added, or follow the prior steps to insert it.
3. Add the expiration and/or warning method to the Window1 class (defined in Window1.xampl.cs). An expiration handling method must take on Boolean parameter which will be set to True if the application has expired, or False if otherwise. Here is an example that shows a warning message during the warning period (on or after the warning date, but before the expiration date) and shows a message and exits the application upon expiration:

   Example

```
private void IsExpired(bool isExpired) {
    if(isExpired) {
       MessageBox.Show("Application is expired!.  Shutting down...");
       System.Environment.Exit(0);
    }
}

private void IsWarning(bool isWarning) {
    if(isWarning) {
       MessageBox.Show("Application will expire soon! ");
    }
}
```

4. Set the *ExpirationNotificationSinkElement* property to **Method**. Set the *WarningNotificationSinkElement* property to **Method**.
5. Set the *ExpirationNotificationSinkName* property to the name of the method to be called when the expiration date has been reached. For this example enter **IsExpired**. Set the *WarningNotificationSinkName* property to the name of the method to be called when the warning date is passed, for this example enter **IsWarning**.
6. Ensure that *Instrumentation* is enabled in the Dotfuscator project's Global Options.
7. Build the solution. This will recompile the application and run Dotfuscator on the output.
8. As before, you can test by setting the warning and/or expiration dates on the Shelf Life attribute to dates in the past.



Summary

Successful development organizations effectively align finite development resources with evolving requirements while simultaneously protecting the value that they create. However, the mismatch between development and business lifecycles, the disconnect between technology creators and business managers, and the loss of control over distributed applications makes application asset management complex and expensive.

Dotfuscator Instrumentation connects the development and business processes freeing developers to focus on core competency while providing an integrated protection and analytic solution able to meet the most demanding business requirements.

This preceding tutorial stepped through a straightforward use case demonstrating essential benefits including:

- **Flexibility** to select any mix of technology at any stage in the development lifecycle
- **Minimal coding** required and only for custom extensions
- **Stability and automation** through deep integration with Visual Studio

However, the inherent limitations of a tutorial format preclude deeper discussions on how this platform can be used to materially improve the value of both your applications and your overall business. For example, Dotufscator Instrumentation can:

- **Improve software sales** cycles increasing win rates, lowering cost of sales and increasing transaction sizes.
- **Extend value-added services** to your end-user organizations generating revenue and competitive differentiators.
- **Improve corporate operations** through application portfolio management.

Email **solutions@preemptive.com** to schedule a meeting to discuss how Dotfuscator can help you protect and increase the value of your application investments.

Index